JYVÄSKYLÄ LICENTIATE THESES IN COMPUTING
18

Antti-Juhani Kaijanaho

The Extent of Empirical Evidence that Could Inform Evidence-Based Design of Programming Languages

A Systematic Mapping Study

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ LICENTIATE THESES IN COMPUTING 18

Antti-Juhani Kaijanaho

The Extent of Empirical Evidence that Could Inform Evidence-Based Design of Programming Languages

A Systematic Mapping Study

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2014

The Extent of Empirical Evidence that Could Inform Evidence-Based Design of Programming Languages

A Systematic Mapping Study

JYVÄSKYLÄ LICENTIATE THESES IN COMPUTING 18

Antti-Juhani Kaijanaho

The Extent of Empirical Evidence that Could Inform Evidence-Based Design of Programming Languages

A Systematic Mapping Study

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2014

Editor
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä

URN:ISBN:978-951-39-5791-9
ISBN 978-951-39-5791-9 (PDF)

ISBN 978-951-39-5790-2 (nid.)
ISSN 1795-9713

Jyväskylä University Printing House, Jyväskylä 2014

GLENDOWER. I can call spirits from the vasty deep.
HOTSPUR. Why, so can I, or so can any man;
    But will they come when you do call for them?

— William Shakespeare's *Henry IV Part 1* (III.1)

ABSTRACT

Kaijanaho, Antti-Juhani
The extent of empirical evidence that could inform evidence-based design of programming languages. A systematic mapping study.
Jyväskylä: University of Jyväskylä, 2014, 243 p.
(Jyväskylä Licentiate Theses in Computing,
ISSN 1795-9713; 18)
ISBN 978-951-39-5790-2 (nid.)
ISBN 978-951-39-5791-9 (PDF)
Finnish summary

Background: Programming language design is not usually informed by empirical studies. In other fields similar problems have inspired an *evidence-based* paradigm of practice. Central to it are secondary studies summarizing and consolidating the research literature. *Aims:* This systematic mapping study looks for empirical research that could inform evidence-based design of programming languages. *Method:* Manual and keyword-based searches were performed, as was a single round of snowballing. There were 2056 potentially relevant publications, of which 180 were selected for inclusion, because they reported empirical evidence on the efficacy of potential design decisions and were published on or before 2012. A thematic synthesis was created. *Results:* Included studies span four decades, but activity has been sparse until the last five years or so. The form of conditional statements and loops, as well as the choice between static and dynamic typing have all been studied empirically for efficacy in at least five studies each. Error proneness, programming comprehension, and human effort are the most common forms of efficacy studied. Experimenting with programmer participants is the most popular method. *Conclusions:* There clearly are language design decisions for which empirical evidence regarding efficacy exists; they may be of some use to language designers, and several of them may be ripe for systematic reviewing. There is concern that the lack of interest generated by studies in this topic area until the recent surge of activity may indicate serious issues in their research approach.

Keywords: programming languages, programming language design, evidence-based paradigm, efficacy, research methods, systematic mapping study, thematic synthesis

Author's address Antti-Juhani Kaijanaho
Department of Mathematical Information Technology
University of Jyväskylä, Finland
PO Box 35 (Agora), FI-40014 University of Jyväskylä
`antti-juhani.kaijanaho@jyu.fi`

Supervisors Professor Tommi Kärkkäinen
Department of Mathematical Information Technology
University of Jyväskylä, Finland

Doctor Vesa Lappalainen
Department of Mathematical Information Technology
University of Jyväskylä, Finland

Doctor Ville Tirronen
Department of Mathematical Information Technology
University of Jyväskylä, Finland

Examiners Doctor Stefan Hanenberg
University of Duisburg-Essen, Germany

Professor Stein Krogdahl
University of Oslo, Norway

ACKNOWLEDGEMENTS

I would like to thank my supervisors, Professor Tommi Kärkkäinen, Doctor Vesa Lappalainen, and Doctor Ville Tirronen, for encouragement, discussions and feedback over the years.

Doctor Stefan Hanenberg of the University of Duisburg-Essen and Professor Stein Krogdahl of the University of Oslo examined this thesis; their comments reflect fairly, in my opinion, both the strengths and the weaknesses of this thesis, and I expect to take their comments into account when I extend the research reported here. My thanks to both. It should be noted that this thesis is published with only minor editorial changes unrelated to the examiners' comments.

The Faculty of Information Technology and the Department of Mathematical Information Technology have generously allowed me to follow my own meandering path in my postgraduate studies alongside my teaching duties. It has taken about a decade to get to this point. The current Dean, Professor Pekka Neittaanmäki, and the current Head of Department, Professor Tuomo Rossi, have particularly offered encouragement, challenges and guidance during this process.

Professor Emeritus Markku Sakkinen, Doctor Hannakaisa Isomäki, Doctor Ville Isomöttönen, Doctor Sami Kollanus, and Mr. Antti-Jussi Lakanen have helped me with discussions, encouragement and critique. Thanks are due to all my other colleagues, as well.

In April 2011, I attended (without a paper to present) the 15th International Conference on Evaluation and Assessment in Software Engineering (EASE) at Durham University, England. I enjoyed the presentations and informal discussions among the participants very much; they also gave me good ideas and inspiration.

The three anonymous reviewers of the EASE 2012 conference gave me useful feedback on an early manuscript detailing the progress of this study.

The interlibrary loan service of the university library responded efficiently to every one of my interlibrary loan requests; I had to make a lot of them during this mapping study. Without them, this thesis could not be what it is now.

My mother, Ms. Maija Tuomaala, with background in philosophy and in education research, gave me useful feedback on many drafts of this thesis and other related manuscripts.

I hereby also thank all my family and friends for encouragement, support and understanding.

NOTE ON ENGLISH USAGE

This thesis, as is customary, has a single author. I find it awkward to use the first person plural (“we”) about work I have done alone, even though it is somewhat conventional. The more usual method of deliberately obscuring agency by using short passive constructions would be, in many cases, inappropriate (while linguistically quite legitimate, see Pullum 2014), as in a systematic secondary study clear indication of who did what is an important part of the audit trail. Thus, like Kitchenham (2010) in her mapping study, I use the singular first person in mine.

In situations where I need to refer to a person whose sex is unknown or immaterial, I will generally use the singular “they” (see e. g. Baranowski 2002; Paterson 2011). Of the many less than ideal options available, it is, in my opinion, the best.

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

How much empirical research there is that could guide a programming language design process to result in a language as useful to the programmer as possible? That is the question I consider in this licentiate thesis, recognizing that such empirical research has not often been taken into account in language design. Answering that question properly required me to conduct an over three years long systematic mapping study, which I now report in this thesis.

There are thousands of programming languages (see e. g. Kinnersley 2001; Pigott 2006), and languages are designed, or their designs improved, all the time (some recent examples: Gerakios, Biboudis, et al. 2013; Kilpatrick et al. 2014; Miller et al. 2014). The designs are generally based on the designers' aesthetics, personal preferences, implementation concerns, and theoretical models. With few exceptions (Myers, Pane, et al. 2004; Cook 2007; Stefik and Siebert 2013), language designers do not consider empirical knowledge regarding programmer behavior and how different language design choices affect it (Hanenberg 2010c; Markstrum 2010).

This is surprising. After all, for instance, the field of psychology of programming is over forty years old (Weinberg 1971; Shneiderman 1980; Hoc et al. 1990; Détienne 2002). Several possibilities to explain this come readily to mind: (1) perhaps the body of knowledge built by the psychology of programming research community is not useful to language designers; (2) perhaps language designers are not aware of such research that is useful; and (3) perhaps language designers, coming mostly from the mathematico–technological background, are intimidated by the inherent uncertainty of behavioral research. I do not investigate these hypotheses in this thesis; I merely offer them as plausible conjectures.

An interesting parallel can be drawn with medicine. There is a huge body of scientific knowledge to draw on regarding the efficacy of various medical interventions. A physician, faced with a patient with particular signs and symptoms, must make a choice as to which diagnosis to make, and what treatment to offer to the patient. One would hope that a physician always chooses the options that have the best support in research. Making that happen is not a trivial undertaking: (1) making sense of medical research literature is a skill separate from the ordinary physician's skills; (2) there is so much of it that a physician is likely overwhelmed; and (3) quite a bit of medical research is unreliable (see e. g. Ioannidis 2005, 2008; Straus et al. 2011).

The now-conventional solution taught to medical students is the paradigm of practice called *Evidence-Based Medicine* (see e. g. Guyatt 1991; Evidence-Based Medicine Working Group 1992; Straus et al. 2011). This is a structured method that an individual physician is expected to apply to resolve uncertainty in handling a particular patient's problem, involving a disciplined search of the research literature, with the aid of secondary and tertiary sources designed for this use. Many other fields have adopted a similar paradigm; most notably, there is *Evidence-Based Software Engineering* (Kitchenham, Dybå, et al. 2004; Dybå, Kitchenham, et al. 2005).

The key conjecture that this thesis is based on is that it might be useful to introduce the evidence-based paradigm to the field of programming language design. Like a physician with a patient, a designer wrangling with a language design often faces uncertainty as to the best way to proceed. Maybe an *Evidence-Based Programming Language Design*[1] paradigm is something that language designers could beneficially adopt.

In this thesis, setting the viability of such a paradigm aside for later study, I deal with a preliminary question:

RQ: What scientific evidence is there about the efficacy of particular decisions in programming language design?

As the phrasing of this question implies, I assume in this thesis that a designer would consult the empirical literature mainly to choose between at least two mutually incompatible design choices, and that the designer is mainly interested in any benefit or hindrance to working programmers caused by making a particular design decision. This leads me to set, for the purposes of this thesis, the following two terminological definitions:

> **Definition 1.** In the context of this study, a *design decision* refers to a particular choice that a programming language designer makes in the course of the design work. In a design decision, the designer chooses one of at least two mutually exclusive choices, each with a different effect on the resulting language design. An archetypal example of a design decision is the choice between static and dynamic typing.

> **Definition 2.** The *efficacy* of a design decision refers, in this case, to the existence and (so far as possible) the magnitude of any benefit (or, negatively, hindrance) to programmers in their programming speed, programming quality, their ability to tackle complex programming problems or other similar matters, broadly construed. Simply put, it is about whether a programmer is helped or hindered in his or her work by a particular design choice, all else being equal.

These definitions prompt the following sub-questions:

RQ1: How much has the efficacy of particular programming language design decisions been empirically studied?

---

1 Note that Stefik, Siebert, et al. (2011) use the phrase "evidence-based programming language" in a different but related sense.

RQ2: Which programming language design decisions have been studied empirically for efficacy?

RQ3: Which facets of efficacy regarding programming language design decisions have been studied empirically?

The following two additional sub-questions are suggested mainly by curiosity, since they are simple to answer while pursuing the previous three questions:

RQ4: Which empirical research methods have been used in studying the efficacy of particular programming language design decisions?

RQ5: How common are follow-up or replication studies, either by the original researchers or by others?

Any study answering any of these five questions is a secondary study, as they deal with the state of the research literature. As is customary in the evidence-based paradigms in the various fields, this secondary study follows a systematic approach. Most systematic secondary studies are either systematic literature reviews (SLRs), which aim to answer specific questions having practical relevance, or mapping studies, which aim to construct a map to the literature. The questions I have posted are fairly broad and are more relevant to researchers than to practitioners. Hence, this is a mapping study.

There are a number of deliberate limits I have set for this study. First, I consider only traditional textual programming languages. Second, I exclude all literature published after 2012. Third, I do not discuss the results of the studies I consider.

Limiting this study only to textual languages means excluding for example visual programming languages and the various integrated development environments, such as Eclipse, from consideration. I appreciate the point made by Myers, Pane, et al. (2004, p. 49) – "features of the programming environment are a crucial part of making a programming language effective and easy to use" – but the scope of this study is large enough even with this exclusion.

I exclude studies published after 2012 mainly because there needs to be *some* cut-off point, so that the literature searches I have made stand some chance of being replicable. I conducted the last searches in early 2013, making the end of 2012 a natural choice.

In this study, I deliberately avoid discussing the results of the studies I have located, as doing so properly would require turning this thesis into a series of systematic literature reviews, one for each topic on which there is relevant research; that would be an enormous undertaking, and one I leave for later. Conversely, dealing with the results in any improper way would be worse than useless, as it could give a false sense of authority to unreliable conclusions. Hence, I avoid them entirely.

I will start by discussing programming languages and their design in Chapter 2. Second, in Chapter 3 I will discuss systematic secondary studies and their methodology in relevant part. Then, I will detail the research design in Chapter 4 and the results in Chapter 5. Finally, in Chapter 6 I will interpret the results and discuss the limitations of this study. Chapter 7 concludes this thesis.

2 PROGRAMMING LANGUAGES AND THEIR DESIGN

In this chapter, I will discuss five topics related to programming languages, based on the literature. First, I need to fix a line of demarcation between programming languages and things that are not programming languages (Section 2.1). Second, I will discuss language classifications (Section 2.2). Third I will explain the conceptual structure conventionally imposed on them (Section 2.3). Fourth, I will examine the key design questions related certain language features of interest (Section 2.3). Finally, I will discuss language design, both historically and the effect of programmer behavior research might have on it (Section 2.5).

2.1 Demarcation

Before one can discuss programming languages, and more importantly, before one can map the empirical literature of use to language design, one must solve the demarcation problem for programming languages: what is, and what is not, a programming language? In the following subsections, I first analyze the concept of language, then the concept of programming, bringing, in the end, the two together to a definition.

2.1.1 Two concepts of language

The *IEEE Standard Glossary of Software Engineering Terminology* (1990), the OED Online (*programming, n.* 2013, compounds), and Dershem and Jipping (1995), as well as perhaps Sethi (1996), adopt similar concepts of language (in this context), based on the idea of combining symbols to communicate ideas. This is a broad concept. It can be argued that the common desktop graphical user interface is a language under this approach: the icons on the screen, the act of pointing at a particular item on the screen using the mouse, and the act of clicking one of the mouse buttons, can be interpreted as symbols, and there are clearly rules that allow combining these symbols to communicate ideas. For example, pointing the mouse at a particular icon and then clicking on the left mouse button twice in rapid succession is a phrase in this language, whose meaning is familiar to all computer users.

In contrast, Sammet (1969) and Fagan (1991) identify a language with the concept of a formal language as that term is used in theoretical computer science, coupled with an intended – and sometimes formally defined – semantics. Gabrielli and Martini (2010) appear to take this position as well, although they do not articulate it. It also underlies the philosophical discussions of programming languages by White (2004) and Turner (2007, 2009); and while Colburn (2000, p. 190) adopts in his philosophical discussion the textbook definition of programming languages (but not of languages) in Dershem and Jipping (1995), he appears to assume that they are formal languages.

The formal language approach (see e. g. Hopcroft et al. 2007) posits that a language is associated with an *alphabet*, meaning a predetermined, finite set of symbols, and is defined by the set of *strings* that the language deems valid; strings being finite (possibly empty) sequences of symbols drawn from the alphabet. In this view, while infinite languages are in practice expressed using a finite description (using one of several formalisms of differing expressive power), the only thing that distinguishes one language from another is their respective sets of valid strings. In the case of programming languages, these strings are conventionally called *programs*, *modules*, or *compilation units*.

Of course, merely knowing which programs are valid in the language is not enough, and thus every programming language, viewed from this formal-language vantage point, has a semantics, assigning an interpretation to every valid program in the language. In the formal point of view, these semantics are typically mathematical functions mapping programs to mathematical objects describing their computational content (*denotational semantics*), mathematical relations between programs and their results (*big-step operational semantics*), or mathematical (transition) relations between states in a special-purpose abstract machine, the states encoding the program and the result, among other things (*small-step operational semantics*). In some cases, particularly in academic publications over the last three decades (e. g. Halpern et al. 1984; Launchbury 1993; Igarashi et al. 2001; Stork et al. 2014), these semantics are specified using mathematical notation and rigor, but the semantics of working languages are usually specified using a natural language such as English. Reynolds (1998) and Kaijanaho (2010) discuss the main techniques of formal semantics of programming languages; the discussion of Java by Gosling et al. (2014) is an excellent modern example of a natural-language description of semantics.

There are two main differences between these two concepts of a language. In the symbols and rules approach, a language is seen first and foremost as a structured concept, built from specific symbols using specific rules, while the formal language approach treats structure as an aid of description, the languages themselves being merely sets of valid strings.

The formal language approach, however, decrees a one-dimensional structure on the utterances allowed by a language: they are built from symbols in a one-dimensional sequence. In principle, any two-dimensional formatting such as line separation and indentation, which are commonly used in programming, are completely ignored as mere presentation issues, although in practice it is possible to treat them, in a limited but meaningful way, by encoding line separation or termination as a symbol in the alphabet and by encoding indentation as one (a tabulation, specifying the indentation for each line independently) or two (indent and dedent, indicating increasing and decreasing levels of indentation, respectively) symbols in the alphabet (see e. g. Marlow 2010; *The Python Language Reference* 2014). In contrast, the symbols and rules approach allows any structure – spatial, temporal, or a combination. As discussed above, a graphical user interface qualifies under this symbols and rules approach, and trying to shoehorn it into a single dimension,[1] which is what is required to make it qualify under the formal language approach, would be more akin to translation to another, quite different language than a mere encoding.

For the purposes of this mapping study, I have decided to adopt the formal language approach, mainly because it offers a fairly clear demarcation line between the traditional programming languages and such things like visual programming languages and integrated development environments.

2.1.2 What qualifies as programming?

The attribute "programming" qualifying the word "language" suggests that not all languages are programming languages. To define the concept of a programming language, one thus needs to consider what "programming" actually means.

Pair (1990) opines that programming is "describing calculations" (p. 11), provided that calculation is understood expansively, including various forms of communication with the external worlds. He also points out that a single program does not describe a single calculation but a "function linking a calculation to each possible input" (p. 10). Détienne (2002, p. 13), based on Pair (1990) and Wirth (1976), characterizes programming as having "two aspects [. . .]: the decomposition of a calculation in order to produce an algorithm and the definition of objects", where by objects she means a generalization of data structures.

Blackwell (2002) characterizes programming in terms of what makes it difficult. The act of programming is separated from the effects of the resulting program in two main ways: firstly, there is temporal separation, as a program is always executed later than it is written, and secondly, there is abstractional separation, as a program is almost always written to be executed more than once, and thus the program must be written to adapt to each new execution context. Blackwell calls them "'abstraction over time' and 'abstraction over a class of situations'" (p. vi). Further, programming requires the use of notation (effectively, a

1 It certainly is possible to do that, as shown by the common implementation approach of representing user actions as a temporal sequence of event descriptions (see e. g. Gettys et al. 2002; *About Messages and Message Queues* 2013), which is simple to encode as a one-dimensional sequence of symbols.

language), and often deliberately uses abstraction to manage complexity.

Blackwell (2002) also advocates phenomenological study of programming in order to characterize the typical programming activity that actually occurs in practice. He further argues that all computer users are programmers: even writing HTML or a complex spreadsheet require temporal separation and often even abstractional separation.

These points lead me to the following conclusion. There is, without doubt, in programming always some computer being instructed. The instruction, which is typically called a *program*, must also be, like both Pair (1990) and Blackwell (2002) note, usable more than once and it must be able to adapt to the context in which it is used; this is typically called its *input*.

2.1.3 Definition

Combining all these threads yields a concept of programming language that can be used as a definition. For the purposes of this mapping study, I will further require that the language is a tool of a programmer, that is, a person who has acquired some skill in and actually engages in the activity of creating programs, whether or not it is their profession (cf. Ko et al. 2011); this also serves to exclude languages meant only as targets for automatically generated code. I will also require, as I am mostly interested in general-purpose languages, that the language must be able to deal with user interaction. This yields the following definition:

> **Definition 3.** A *programming language* is a formal language (that is, a set of strings) with an associated (implicit or explicit) semantics, intended for use or is used[2] by programmers to construct reusable instructions (a *program*) for a computer to perform a specific task in response to, or in light of, external input, possibly including user interaction.

I should note that this definition is intended (and I have interpreted it, in the course of this study) to exclude such languages as SQL and HTML, for the lack of ability to deal with user interaction, as well as visual languages, for not being a set of strings.

2.2 Classifications

There are four commonly mentioned classifications of programming languages: language levels, generations, paradigms and the systems programming language versus scripting language dichotomy. All four occur in the studies included in this mapping study.

[2] This grammatical error was introduced in the first version of the protocol that carried this definition and went uncorrected in all supporting materials during the study. I retain the exact phrasing, including the error, for audit trail purposes.

2.2.1 Language levels

Every computer has a native language (called a *machine language*). The machine languages of many of the earliest stored-program computers, different for each machine, were directly readable and writable via the native character set of the machine by their human programmers; the language of the Cambridge University computer EDSAC, at least, was even somewhat mnemonic (Wilkes et al. 1951; *Programming for the UNIVAC Fac-Tronic System* 1953, p. 24-25; Campbell-Kelly 1980a,b; Wheeler 1992; Koss 2003). Other computers (particularly modern ones) use a machine language that requires a separate coding step from the machine-language programmer's notes to machine language, and a decoding step if the program already stored in a computer is to be read by someone. All machine languages, even the alphanumeric machine languages of computers like the EDSAC, require detailed bookkeeping on the part of the machine-language programmer to keep track of memory addresses, and even the slightest changes to the program require detailed manual recomputation (see e. g. Koss 2003, p. 52).

The coding and bookkeeping required to program with a machine language are tedious mechanical processes, and thus good candidates for automation. Programming techniques required to produce *assemblers* that took a readable but extremely detailed description of a machine-language program and converted it into machine language were developed by the early 1950s. The language understood by a particular assembler is called an *assembly language*.

Practically all programs require the computation of nontrivial arithmetic; for example, to access the ith element of an array that starts at address a and whose elements are b bytes long (including any padding) requires the computation of $a + (i - 1)b$. This operation is found in essentially all programs. In machine and assembly languages, the programmer is required to sequence the computation and keep track of storage for the intermediate values by hand. The programmer is also required to juggle the extremely limited number of registers, and to take into account the numerous special cases and warts that a machine language typically provides to a programmer.

High-level languages are programming languages that abstract away such details. The programmer may write arithmetical formulas directly in their program, without worrying about sequencing of the arithmetic and intermediate value storage. The programmer may pretend the machine is more regular than it is, not caring about the limited number of registers and other technical warts of the machine. A high-level language also hides all details concerning address calculation from the programmer, who writes only in terms of symbolic names. Most high-level languages are sufficiently abstracted from the details of a particular machine that programs written in them can be portable.

This definition is largely equivalent to that given by Sammet (1969, p. 8–11) and the *IEEE Standard Glossary of Software Engineering Terminology* (1990, p. 37). Some authors exclude languages like C, mostly because they do not provide as much abstraction capability as many other commonly used languages (see e. g. Graunke et al. 2001; Lin and Blackburn 2012).

Low-level languages are languages that do not qualify as high-level languages; that means most machine languages and assembly languages, but there have also been other low-level languages as well (e. g. Crary and Morrisett 1999). Note that some low-level languages do not qualify as programming languages as I have defined them earlier, because they are only intended for use and only used as code-generation targets.

2.2.2 Generations

The most commonly mentioned *programming language generations* are the following (see e. g. Martin 1985; *IEEE Standard Glossary of Software Engineering Terminology* 1990; O'Regan 2012, p. 121–124; Rawlings 2014, p. 33):

1. The first generation consists of machine languages.
2. The second generation consists of assembly languages.
3. The third generation consists of high-level languages (in the expansive sense that includes e. g. C).
4. The fourth generation typically refers to high-level languages that provide various facilities to process large masses of data (such as databases) with a small amount of programming effort; Martin (1982, p. 28), for example, requires a language to be at least an order of magnitude more productive than COBOL, a quintessential third-generation language, to belong in the fourth generation, while three years later he merely states that such languages "permit some applications to be generated with one order of magnitude fewer lines of code than would be needed with COBOL, PL/I, ADA, or the like" (Martin 1985, p. 4–5).
5. The fifth generation comprises languages, like Prolog, that allow the programmer to specify constraint-solving problems in a relatively natural manner without having to specify a constraint-solving algorithm.

A key weakness of the generation concept is that it implies a rough temporal sequence: one would expect all languages of the same generation to be roughly contemporaneous, and the generations to follow each other in an orderly fashion, albeit with some overlap. Yet, assembly languages developed concurrently with the early high-level languages, not before them, and new assembly languages have appeared decades after high-level languages became commonplace. Finally, none of the five generations have yet perished.

Worse, this is not the only classification of languages by generations. For example, Wegner (1990, p. 19–20) identifies the first three generations with particular years, with the first occurring on 1954–1958 and including languages like the original FORTRAN, the second 1959–1961 including FORTRAN II, ALGOL 60, and COBOL, and the third 1962–1969, including PASCAL and SIMULA; he does not acknowledge any later generations, instead calling the years 1970–1979 (e. g. Ada and Smalltalk) "[t]he generation gap" and assigning the years 1980–1989, which take him to the year on which he was writing, to "[p]rogramming language paradigms".

2.2.3 Paradigms

The third well-known categorization is, in fact, the concept of *paradigm*. In ordinary English, the word means (*paradigm, n.* 2014, sense 1)

> "A pattern or model, an exemplar; (also) a typical instance of something, an example."

In 1962, Kuhn (1996, p. 10) famously appropriated the word to describe

> "accepted examples of actual scientific practice [that] provide models from which spring particular coherent traditions of scientific research"

Explicitly citing Kuhn, Floyd (1979) introduced in his Turing award lecture the idea of *paradigms of programming*, by which he meant particular ways to organize programs, such as structured programming and dynamic programming. Unlike Kuhn,[3] whose paradigms were incommensurable and fundamentally incompatible with each other requiring a scientific revolution to effect a paradigm shift, Floyd urged programmers to "expand [their] repertory of paradigms" (p. 457, emphasis deleted). The phrase has been mentioned, apparently with this meaning, even before Floyd's lecture, but only in passing (Goldstein and Sussman 1974, p. 13; Davis 1977, p. 47).

In the following decade and a half, a number of programming paradigms, particularly focusing on high-level issues, became popularly accepted. Ambler et al. (1992) identified a number of them: imperative, object-oriented, functional, asynchronous parallel, synchronous parallel, transformational, logic, form-based, dataflow, constraint, and demonstrational. They noted, further, that many programming languages reflect particular paradigms, so much so that they are "often hard to distinguish from the paradigm itself" (p. 28).

Reflecting that comment, in recent usage, programming paradigms are generally taken as programming *language* paradigms: categorizations of programming languages based on language features they possess, originally inspired by the Floyd-style programming paradigms that those features were designed to support. Van Roy (2009) argues for a taxonomy of 27 modern programming (language) paradigms, including the well-known ones: imperative programming, functional programming, (sequential) object-oriented programming, and logic programming. The *Computing Curricula 2001* (2001, p. 113) recommended that five paradigms be surveyed briefly in a computer science undergraduate curriculum: procedural, object-oriented, functional, declarative, and scripting. The *Computer Science Curricula 2013* (2013, p. 156) recommend, without invoking the word "paradigm", teaching object-oriented programming, functional programming, event-driven and reactive programming, and logic programming, among many other things.

In this study, despite their disadvantages, programming paradigms do play a role, chiefly because the primary studies I have studied in this mapping study employ them. The following programming paradigms are of special interest to

3 Incidentally, Priestley (2011) has identified a true Kuhnian paradigm in programming language research: ALGOL.

this study, defined by their main program composition or decomposition approaches:

- *Imperative* programming decomposes programs into a sequence of steps that must be followed without deviation except when a step explicitly calls for an altered flow of control (such as a conditional or a loop). Some authors have called this procedure-oriented or procedural programming (Katz and McGee 1963; Sammet 1969, p. 19–20; Leavenworth and Sammet 1974), but I reserve that label to the another paradigm (as does e. g. Simmonds 2012).
- *Procedural* programming decomposes programs into procedural or imperative subprograms which are invoked by name, may take parameters, may return a value and may have side-effects (see e. g. Simmonds 2012).
- *Structured* programming is an umbrella term encompassing a number of programming paradigms related to imperative and procedural programming, particularly stepwise refinement (decomposing a program into an imperative program using calls to fictional subprograms to delegate non-obvious tasks for later programming, followed by doing the same to each of the fictional subprograms), the use of a restricted set of control-flow constructs (sequencing, selection, and loop), and the avoidance of goto statements (Weiner 1978).
- *Object-oriented* programming decomposes programs into objects possessing identity, state and behavior which communicate by invoking each others' methods and which may be related by some incremental modification device such as class inheritance (Wegner 1987; Stroustrup 1988; Taivalsaari 1993, 1996). Support for classes is common but not a requirement.
- *Functional* programming composes programs mostly from existing functions using functionals (higher-order functions) (see e. g. Hughes 1989). Purity (lack of side-effects) and lazy evaluation of the functions are common but not required.
- *Aspect-oriented* programming decomposes a program in more than one way, encapsulating non-principal decompositions into aspects that interact with the principal decomposition and each other at particular join points (Kiczales, Lamping, et al. 1997).

Note that these are not exclusive language categories, as many languages qualify for more than one. For example, AspectJ (Kiczales, Hilsdale, et al. 2001) is an aspect-oriented language that encourages object-oriented programming for the principal decomposition. Almost all procedural and object-oriented languages are also imperative languages.

A third categorization, essentially another pair of paradigms, was introduced by Ousterhout (1998). He distinguished *system programming languages*, by which he meant the traditional high-level languages such as Pascal, C, C++, and Java, from *scripting languages*, such as the Unix shells, Perl, and Tcl. The latter term was not his invention, but he gave it a specific meaning. System programming languages are, according to him, designed for writing software from the ground up, while scripting languages take for granted that there is existing software to be glued together in order to create new useful software. The former languages are typically compiled and have static type systems, while the latter languages are often interpreted and use dynamic type systems. Showing the relevance of the distinction, Spinellis (2005) and Loui (2008) debated the viability of scripting languages, but neither questioned the category itself.

The concept of language paradigms is widely accepted but, I think, problematic. Krishnamurthi (2008) argues that teaching language paradigms is "a misguided attempt to follow the *practice* of science rather than its *spirit*" (p. 81, emphasis in the original); similarly, Stroustrup (2014, p. 11) considers the idea of a paradigm "pretentious", preferring instead to say that a language "provide[s] support for programming styles" (p. 10). I largely agree; while the idea that a language is more similar to certain languages than some others is intuitively obvious, trying to formalize it into some sort of a taxonomy likely does more harm than good, as it tends to create factions centered around particular languages. The taxonomy proposed by Van Roy (2009) makes more sense, as it is centered around categorizing language features, not languages *per se*, but it should probably not be called a taxonomy of paradigms. The idea of a programming style (or, in Floyd's terminology, paradigm) makes sense so long as, like Floyd (1979) and Stroustrup (2014), one recognizes that they are not mutually exclusive.

2.3 Conceptual structure

I have already defined a programming language (Definition 3 on page 19) as having structure: it is a set of strings (*programs*) with an associated semantics. There is traditionally, however, a more detailed conceptual structure of programming languages, based on the typical structure of a compiler or an interpreter, that is almost universally used to discuss them: a language is said to have both a lexical and a syntactic structure, and both static and dynamic semantics; moreover, the adjectives "static" and "dynamic" are widely used to classify the properties of a language. In this section, I will review these concepts, as background for the rest of this chapter and the mapping study.

Programming languages are typically formal languages of some standard alphabet, usually ASCII ("American Standard Code for Information Interchange" 1963) or Unicode (Allen et al. 2013). The *lexical* structure of a programming language assigns to each program of the language a sequence of *lexemes* (sometimes called *tokens*), which usually are non-overlapping substrings of the program often separated by non-significant characters (usually whitespace), and categorizes lexemes into *lexical categories* (or *token types*); it also rejects some strings of the alphabet as lexically erroneous.

The *syntactic* structure of a programming language assigns to each program (typically treating it as a sequence of lexemes and ignoring the details of each lexeme beyond its lexical category) a *syntax tree* describing the hierarchical structure of the program. It also rejects some putative programs as syntactically erroneous.

The syntax of a programming language usually comes in two varieties: the *concrete syntax*, which defines *concrete syntax trees*, is strictly tied to the lexemes that make up programs. In contrast, *abstract syntax*, which defines *abstract syntax trees* (or *AST*s), elides details that are necessary for an unambiguous syntactic analysis of programs but unnecessary from a semantic point of view, such as the presence of parentheses and the concrete operator signs in an arithmetic expression (the AST will use other means than remembering the concrete character to indicate which operation is needed). A language that defines both will usually also define (often implicitly) the relationship between actual lexeme sequences to abstract syntax trees.[4]

The *semantics* of a language assigns to each program (typically treating it as an abstract syntax tree) a meaning. It is generally defined recursively, by giving a meaning for each possible subtree of an abstract syntax tree and deriving the semantics of larger trees in terms of the meaning of its subtrees. This meaning, in particular, defines the behavior of the program for each permissible execution context (including any input).

The semantics of a programming language may *reject* some programs, in either all or some execution contexts, and it may be *undefined* for some programs in some execution contexts. The difference is practical: a programmer can expect to be told of a rejection but cannot expect anything with respect to programs with undefined semantics. In any case, a program that is rejected or has undefined semantics is said to be *semantically erroneous*. A language that has no undefined semantics is sometimes called *safe*, although some authors additionally require that the abstractions that the language provides do not leak (see e. g. Pierce 2002, p. 6–8).

The precise boundaries between lexical, syntactic, and semantic structure is malleable. They are, after all, only aids for language definition and analysis, not laws of nature. One particular distinction between syntax and semantics is, however, worthy of note. The description of the first language to use formal grammar in its definition, Algol 60, discussed each language feature in at least three parts: first syntax, then examples, then semantics, followed by additional subsections as necessary (Naur et al. 1960). The syntax descriptions contained only context-free grammars, using the then-new Backus–Naur Form (BNF), and the semantics included statements like the following (p. 302):

> "The same identifier cannot be used to denote two different quantities except when these quantities have disjoint scopes as defined by the declarations of the program"

Griffiths (1975, p. 83), writing for a 1972 advanced course on software engineering, articulated a difference between *static semantics*, "that part of the semantics which does not depend upon the execution of a program", like the Algol passage

4 Strictly speaking, abstract syntax is truly abstract and does not involve actual trees. For the purposes of this mapping study, that is a bit too abstract. The tree metaphor is close enough, especially considering that abstract syntax representations of programs are often, in practice, tree data structures. Abstract syntax, in the truly abstract sense, was introduced by McCarthy (1996) in 1962; an elegant mathematical formulation based on universal algebra was given by Gougen et al. (1977).

I quoted, and *dynamic semantics*. Practically speaking, he pointed out, static semantics describes the behavior of the language compiler, and dynamic semantics the behavior of the machine-language program it generates. He did not claim to have invented these terms, but he does not attribute them to anyone else either, and I have not been able to find any earlier source for them.

The distinction has been frequently used in the literature up to the present day (e. g. Gerakios, Papaspyrou, et al. 2014; Slepak et al. 2014); a more recent formulation has static semantics defining well-formedness, "a kind of (context-sensitive) syntax", while "dynamic semantics is about computation" (Mosses 2001, p. 167, emphasis deleted; see also Gabbrielli and Martini 2010, p. 40). Koster (1974) and Meek (1990), however, make a case that static semantics is a misnomer, belonging properly under syntax. Sakkinen (1992) and Harel and Rumpe (2004), among others, adopt a similar point of view. Harper (2014) takes a different approach: he labels lexical and syntactic structure together with static semantics collectively as *statics*, calling dynamic semantics *dynamics*.

More generally, *static* is often used as an adjective meaning roughly 'independent of any particular execution of the program', and *dynamic* as meaning 'pertaining to or depending on a particular execution of the program'; the adverbs *statically* and *dynamically* are used with similar meanings.

These concepts are offered here mostly as background, which is used freely in later parts of this thesis.

2.4 Development of certain features

In this section, I will review the key design options available on two language features, conditional statements and typing. The review is partly conceptual, partly historical; the latter partly to give the necessary historical context to certain studies included in the results of this mapping study, and partly to acknowledge the contribution of specific people in the development of these features.

These two features were chosen because they are prominent in the results of this mapping study. Additionally, the design choices involving conditionals that have been investigated in the included studies include several now rare options, and they need to be introduced. Further, in the case of typing, there is no consensus on what it encompasses and what words are used to name the key concepts; I thus need to introduce the competing traditions and establish specific definitions for the purposes of this mapping study.

2.4.1 Conditionals

All programs need to be able to choose between two or more different execution paths based on the current state of the program at the time of the choice. Low-level languages usually offer the ability to jump to a specified location in the program if a particular quantity is negative, zero, or positive. A similar approach was taken in the early FORTRAN (Backus et al. 1956, p. 18), where an IF statement like `IF (A-B) 10,20,30` jumps to the line labeled `10` if the expression `A-B` evaluates to a negative value, to the line labeled `20` if the expression evaluates to zero, and to the line labeled `30` if the expression evaluates to a positive value. This style of a conditional was later labeled an "arithmetic IF", to distinguish it from the "logical IF" (*FORTRAN IV Language* 1963, p. 12) statements like `IF (A.LE.B) GO TO 10` which jumps to the line labeled `10` if `A` is strictly less than `B`, and proceeds to the statement following the IF otherwise (almost any statement could replace the `GO TO`).

The International Algebraic Language or IAL (Perlis and Samelson 1958), which is better known under the name ALGOL 58, included an *if* statement much like the later "logical IF" of FORTRAN IV. The *if* statement made the statement following it conditional. For example, in *if* $a > 0\,;b := a \times b$, the multiplication and assignment are performed only if a is positive. From a language structure point of view, the *if* and the assignment were, in the IAL, separate statements, the *if* merely affecting the assignment as a side-effect, instead of the assignment being a substatement of the *if*, like in modern high-level languages. However, the IAL also allowed the formation of compound statements by enclosing a sequence of statements in the parenthetical keywords *begin* and *end*; this made the IAL *if* much more powerful than the later FORTRAN IV logical IF, by allowing a single *if* statement control more than one statement at the same time without resorting to any *go to* statements.

Based on a proposal by Green et al. (1959), ALGOL 60 (Naur et al. 1960, 1963) included an enhanced **if** statement. First, the statement that the **if** controls is a substatement of the **if**, separated from the Boolean expression not by a semicolon but the keyword **then**; second, that substatement may be optionally followed by the keyword **else** followed by another substatement. In ALGOL 60, it was thus possible to write, for example

$$\textbf{if}\ a > 0\ \textbf{then}\ b := a \times b\ \textbf{else}\ b := 1$$

meaning that b is assigned $a \times b$ if a is positive, and 1 otherwise. Of course, since ALGOL 60 included conditional expressions (as proposed by McCarthy 1959), the same operation could have been written as

$$b := \textbf{if}\ a > 0\ \textbf{then}\ a \times b\ \textbf{else}\ 1$$

The ALGOL 60 style **if–else** construct is famously susceptible to a grammatical ambiguity: what is the value of x after the ALGOL 60 style statement

$$x := 0\,;\textbf{if}\ a > 0\ \textbf{then}\ \textbf{if}\ a > 2\ \textbf{then}\ x := 1\ \textbf{else}\ x := 2$$

when the value of a is 1?[5] A number of solutions were proposed in the years following the publication of ALGOL 60 (see Abrahams 1966), including revising

5 This particular example is forbidden by the ALGOL 60 grammar, but many later languages allow statements of this kind. Even ALGOL 60 is susceptible to this problem, but the examples are more complex (see e. g. Kaupe 1963).

the grammar to remove the ambiguity, declaring a disambiguation rule verbally, and making the **else** mandatory. Abrahams (1966) himself proposed an elegant grammar revision, which is now a textbook solution (e. g. Aho et al. 2007, p. 210–212). ALGOL 68 (Wijngaarden et al. 1976), in which there was no distinction between expressions and statements, introduced another solution: requiring that a keyword is used to end every **if** expression; in bold-style reference-language ALGOL 68, the keyword was **fi**, but many other languages have opted for other keywords. Some modern languages, like Perl 5 (*perlsyn* 2014) instead have made it mandatory to use the equivalent of **begin**–**end** bracketing in a conditional statement.

Sime et al. (1999), originally published in 1973, called the FORTRAN logical IF style conditionals JUMP, and the ALGOL 60 conditionals NEST. Sime et al. (1977) named a variant of NEST, in which **begin** and **end** are mandatory, NEST-BE, and they also introduced a new variant, which they called NEST-INE (if–not–end). In it, there is a mandatory phrase for ending a conditional statement: the keyword **end** followed by a repeat of the condition. Additionally, in NEST-INE, the keyword **else** is replaced by a phrase consisting of the keyword **not** followed by a repeat of the condition. The previous ambiguous example might be rendered in a NEST-INE variant of ALGOL 60 in either of the two following ways, reflecting the two interpretations of the original example:

$$
\begin{array}{ll}
\begin{array}{l}
x := 0; \\
\textbf{if } a > 0 \textbf{ then} \\
\quad \textbf{if } a > 2 \textbf{ then} \\
\qquad x := 1 \\
\quad \textbf{end } a > 2 \\
\textbf{not } a > 0 \textbf{ then} \\
\quad x := 2 \\
\textbf{end } a > 0
\end{array}
&
\qquad
\begin{array}{l}
x := 0; \\
\textbf{if } a > 0 \textbf{ then} \\
\quad \textbf{if } a > 2 \textbf{ then} \\
\qquad x := 1 \\
\quad \textbf{not } a > 2 \textbf{ then} \\
\qquad x := 2 \\
\quad \textbf{end } a > 2 \\
\textbf{end } a > 0
\end{array}
\end{array}
$$

Embley and Hansen (1976) and Embley (1978) defined a new control structure unifying iteration and conditionals, the *KAIL selector*. The following is an example given by Embley (1978, p. 200, direct quote):

$$
\begin{array}{l}
x \leftarrow \mathit{rand}(25); y \leftarrow \mathit{rand}(25); \\
\quad \mathit{comment}\ \text{set } x \text{ and } y \text{ to random integers in } [1, 25]; \\
\mathit{write}\ \text{What is } \langle(x)\rangle + \langle(y)\rangle; \\
[\mathit{accept}\ \text{reply}; \mathit{if}\ \text{reply} \\
\quad | = x + y:\ \mathit{write}\ \text{Very good; correct\_count} \leftarrow \text{correct\_count} + 1; \\
\quad | = x * y:\ \mathit{write}\ \text{Add, don't multiply;}\ \mathit{again}; \\
\quad | > x + y + 10:\ \mathit{write}\ \text{No, that's more than 10 too much;}\ \mathit{again}; \\
\quad |\, \text{else}:\ \mathit{write}\ \text{No, try again;}\ \mathit{again}; \\
];
\end{array}
$$

This program fragment picks two random numbers and tests whether the user can correctly add them together. The KAIL selector consists of the square brackets and everything in between; it starts with an initialization command ("*accept* reply"), then evaluates the discriminator ("*if* reply") and picks the first of the multiple alternatives that results in a true test result. Within each alternative, the "*again*" statement directs execution to go back to the beginning of the selector, much like a `continue` statement in C or Java inside a loop.

Many currently popular languages follow the NEST model, with only cosmetic changes. For example, C (Ritchie 1974; Kernighan and Ritchie 1978, 1988; *Information Technology – Programming Languages – C* 2011), and languages descended from it, like Java (Gosling et al. 2014) and C# (*Information Technology – Programming Languages – C#* 2006), require an opening parenthesis immediately after the `if` keyword, replace the `then` keyword with a closing parenthesis, and replace the compound-statement-bracketing keywords `begin` and `end` with the curly braces `{` and `}`, respectively. As already mentioned, Perl 5 (*perlsyn* 2014), which is a descendant of C, follows the NEST-BE style, albeit using the C-style cosmetics. Many of these languages also allow the logical IF, or JUMP, style, although for example Java (Gosling et al. 2014) forbids it (by not providing a `goto` statement). I am not aware of any current high-level languages that allow the NEST-INE style, the KAIL selector, or, apart from FORTRAN, the arithmetic IF.

2.4.2 Types

Integers and floating-point numbers have incompatible representations and use different machine-language instructions to do arithmetic. In arithmetic formulas, a *sine qua non* of high-level programming languages, this distinction is absent. The problem for language designers is obvious: how does a compiler know whether to use `ADD` or `FADD` (to use the modern IA-32/64 instruction names) to compile $a + b$?

FORTRAN (Backus et al. 1956) used a lexical solution: integer expressions consisted of integer constants (easily lexically distinguished from floating-point constants) and integer variables (distinguished by starting with `I`, `J`, `K`, `L`, `M`, or `N`) and were thus readily distinguishable from floating-point expressions.

The IAL (Perlis and Samelson 1958) and its successor ALGOL 60 (Naur et al. 1960, 1963) retained the idea of lexically distinct integer constants but introduced the idea of a *type declaration*: a phrase within the program declares a particular variable to be an integer, a real (floating-point) number, or Boolean, within the whole program or only inside a particular block. Unlike many later languages, the two ALGOL languages did not regard the arrayness, functionness or procedureness of a variable to be a part of its type; after all, the use of an identifier as an array, function, or procedure name was readily syntactically apparent.

From these two early examples, it is apparent that, as Strachey (2000, p. 35)[6]

[6] Strachey wrote this paper in 1967 based on lectures he gave in the International Summer School in Computer Programming in Copenhagen in August 1967; the proceedings the paper was intended for never appeared, but the paper was widely circulated in manuscript

noted, a *type* (in this basic sense) has two facets: it determines the *representation* of a value and the choice of interpretation for operations applied to it (nowadays called *overloading resolution*). The need for the second facet springs from the first facet: if integers and floating-point numbers had the same representation, they could be uniformly added, multiplied, and so forth, and there would be no need to choose between multiple interpretations.

It is, of course, possible to have structure in data. A very early language, FLOW-MATIC (*UNIVAC FLOW-MATIC Programming System* 1958), separated data description (given in separate data description forms which were subsequently typed on magnetic tape for input to the compiler) from the algrithm description; according to Sammet (1969, 1981), it was the first language to do so.[7] Directly influenced by FLOW-MATIC, a rather powerful facility for describing structured data was included in the Common Business Oriented Language COBOL (*COBOL* 1960), and from there borrowed to at least PL/I (Radin 1981; Shneiderman 1985). In none of these languages was data structuring considered a typing issue, however.

Hoare (1965, 1966), aware of COBOL and inspired by Ross and Rodriguez (1963) and McCarthy (1964), proposed for the next version of ALGOL[8] a facility for the programmer to define new types, the values of which are references to mutable records of named and typed fields, all records of the same type sharing the same list of field names and types. Among their other influence, Hoare's records inspired changes to an ALGOL-derived language, SIMULA (Dahl and Nygaard 1966), developing into the classes that are a central part of programming in languages like Java and C# (Krogdahl 2005).

Records, both in the COBOL sense and in the Hoare sense, determine the representation of a data item and overloading resolution for the operations applied to it, just like the types of early FORTRAN and ALGOL 60. Records, however, add a further complication: there are operations that make no sense applied to them (for example, computing the sum of two symbol table entries in a compiler), and the operations that do make sense for records do not make sense applied to integers or floating-point numbers. Thus, types in the record era clearly have a third function: they define *interfaces*, that is, what operations are allowed.

All of the languages thus far mentioned treat types as static notions. After all, in both FORTRAN and ALGOL 60 the reason types exist at all is to provide the compiler with compile-time (that is, static) information to direct the compilation process. However, if one instead inverts this relation, and takes as a premise that integers, floating-point numbers, and records form types that determine representation, overloading resolution, and interface, it becomes apparent that there is

---

form in the decades before its posthumous formal publication in 2000 (Mosses 2000).

7 Curiously, Knuth and Trabb Pardo (2003), in their well-regarded survey of pre-ALGOL languages, dismissed FLOW-MATIC summarily, in barely two paragraphs and with a minimal example, noting that it "had a significant effect on the design of COBOL" (p. 73); they did not even mention its data structuring capability.

8 The incorporation of a feature in the ALGOL development is significant mainly because many current languages derive from proposals floated during the 1960s for the next version of ALGOL.

nothing compelling types to be static. After all, one can store enough information in each runtime value to determine what representation it uses, how overloading is to be resolved and what the interface of the value is. Indeed, a number of languages leave the type concept dynamic, starting from LISP (McCarthy 1960), continuing through for example BASIC (Kurtz 1981) and Smalltalk (Goldberg and Robson 1983), and including such recent languages as Perl,[9] Python,[10] JavaScript (see e. g. Mikkonen and Taivalsaari 2008), and Ruby.[11]

At this point, let me define some common terms. A *type error* is synonymous with the violation of an interface: an operation is applied to a value or object for which the operation is not allowed (often because it does not make sense). *Type checking* refers to language-mandated checking for type errors. A *type system* is the part of a language that defines what types exist (or can be created by the programmer), what the type errors are, and what sort of type checking is mandatory. *Static typing*, *static type system*, and *static type checking* refer to type systems in which types and type errors are static notions, with type checking expected to be performed before a program is allowed to execute. *Dynamic typing*, *dynamic type system*, and *dynamic type checking* refer to type systems in which types and type errors are dynamic notions, and type errors are checked for during each execution, concentrating only on type errors that actually are about to occur during the execution. Sometimes, a type system is characterized as *strong* or *weak* based on how well it detects type errors. Definitions like these are fairly commonly accepted (see e. g. Sheil 1981; Cardelli and Wegner 1985; Allende et al. 2013; Hanenberg, Kleinschmager, Robbes, et al. 2013; Turner 2013; Harper 2014), but, as I will discuss below, they are not accepted by all authors.

While representation and overloading resolution are tightly coupled, the same cannot be said for interfaces. For example, the interface for a datum representing an arithmetic expression in a calculator or language interpreter does not necessarily depend on whether the datum is represented as a string of characters or as an abstract syntax tree (represented typically as a graph of records). It is therefore not a surprise that a number of researchers have advocated splitting representation and overloading from interfaces (e. g. Liskov and Zilles 1974). It is, of course, a central idea in object-oriented programming, and dynamic typing in general.

There is a second tradition of types, now over a century old, which started to mix with the programming language type tradition in the late 1960s and is now dominant in academic research of programming language type systems (Pierce 2002). The tradition began in response to the late 19th Century mathematics, which had delivered a number of new strange results and paradoxes and thus shaken the mathematicians' confidence in their methods. The simplest of the new paradoxes is due to Russell (see e. g. Irvine and Deutsch 2013): is the set of all such sets that are not an element of themselves an element of itself? These developments prompted the building of firm foundations based on logic.

---

9 http://www.perl.org/

10 https://www.python.org/

11 https://www.ruby-lang.org/

Russell himself proposed the *theory of types* (Russell s.d. Appendix B, 1908; Whitehead and Russell 1910; for a recent reformulation, see Kamareddine et al. 2002). Its key concept was a propositional function – a higher-order logical formula, whose free variables were interpreted as parameters of the propositional function. Each variable, whether free (and hence a parameter) or bound by a quantifier, was required to take values of one type only, the type being freely choosable for each variable. All individuals belonged to one type common to them all. All propositional functions sharing the same number and type of parameters also shared a type. Type thus identified whether a variable could take individual or function values, and for the latter, the number and typing of its parameters. The type did not identify a function's result, because all functions were propositional, meaning that they all resulted in either "true" or "false".

Additionally, Russell's theory of types required each variable to restrict the values it takes to a single *order*, which was identified by a finite ordinal. The zeroth order consisted of individuals and propositional functions containing no variables (whether free or bound). The order of a propositional function containing at least one (free or bound) variable was one greater than the maximum of the orders of the variables it contains. Thus, a function with no bound variables taking one individual argument was a *first-order function*; if it, however, used a variable of the first order (either supplied as another parameter or bound by a quantifier), it would have been a *second-order function*.

A formula of Russell's theory was required to be free of both type violations and order violations. This two-pronged approach gave it the name the theory is today known: the *ramified theory of types*. The reason for the use of both types and orders was fairly technical, which I will not discuss here. The deramification of the theory, meaning the removal of orders from it, was suggested by at least Chwistek (1922, 1925), Ramsey (1926) and Hilbert and Ackermann (1928, p. 114–115); it was equally based on technical reasons related to the development of logic. The deramified theory, considering only types and ignoring orders, acquired the name *simple theory of types*, as it was significantly simpler than the ramified theory.

The simple theory of types (sometimes called the theory of simple types or simple type theory) is now better known in the formulation originally given by Church (1940). Instead of having the parameters of a propositional function be implicitly defined by its free variables, he introduces (based on earlier non-typed work, see Church 1932, 1941) a quantifier-like binder, written in modern notation λxt, which converts the term t into a function of x; he also introduces a corresponding operation tu, which supplies the argument u to the function t. The simple theory of types, in this formulation, requires a function type to specify both the parameter type T and the result type U, written in modern notation as $T \to U$. A cleaned-up version of Church's simple theory of types has been standard material in the theory of programming language types for some time now under the name *simply typed lambda calculus* (Cardelli and Wegner 1985; Barendregt and Hemerik 1990; Pierce 2002; Cardelli 2004).

The relevance of typed logics to programming languages became apparent rather slowly. While McCarthy (1960) had modeled some aspects of LISP on the (untyped) lambda calculus, and while Landin (1965) had pointed out a close correspondence between ALGOL 60 and the (untyped) lambda calculus, neither of them considered the simply (or otherwise) typed lambda calculus. It appears Morris (1969) was the first to explicitly investigate the simply typed lambda calculus (which he appears to have independently rediscovered) in the programming language context. Reynolds (1974) extended typed lambda calculus to support basic parametric polymorphism, unaware that the logician Girard (1971, orally presented in 1970) had done the same some years earlier; this type system is variously called (taxonomically) the second-order lambda calculus, (following Reynolds) the polymorphic lambda calculus, or (after its accidental name in Girard's paper) System F. Milner (1978), unaware of earlier very similar work by Hindley (1969), defined a restricted variant of the Girard–Reynolds second-order lambda calculus in which no type declarations were required; this system is now called the Hindley–Milner type system, and it is the basis of the type systems of ML and Haskell. By the time Cardelli and Wegner (1985) and Reynolds (1985) published their reviews, typed lambda calculus and related formal systems appear to have been a part of the standard research toolset – although not the main tool, as it is now (Pierce 2002; Cardelli 2004). Incidentally, there is a repeated pattern of logical concepts being rediscovered by programming language type system researchers, unaware of the earlier work (Wadler 2000).

The logicians' concept of type systems, reinterpreted in the context of programming languages, is exclusively static. The express purpose of type systems in logic is to exclude syntactically valid expressions from semantic consideration. From a logician's point of view, a language that does not have a static type system is untyped, not dynamically typed. This has lead some authors (such as Pierce 2002; Cardelli 2004; Trancón y Widemann 2009) to declare that even in the programming language context, types and type checking are exclusively static concepts, and to discourage the use of terms like dynamic typing. I decline to adopt that point of view for the purposes of this mapping study.

2.5 Design

In this section, I will look at programming language design, first as a question of historical practice, then reviewing the influence of research on programmer behavior on it, and finally introducing the idea of Evidence-Based Programming Language Design.

2.5.1 Historical practice

A number of opinion essays on language design have been written over the decades. For example, Hoare (1989), gave a number of "hints" to programming language designers, on both overall design goals and on specific features, mostly argued informally; they included the following five "catch phrases" he intended to summarize "objective criteria for good language design" (p. 197): "simplicity, security,[12] fast translation, efficient object code, and readability". It is curious that Hoare calls them objective criteria, when reasonable people disagree on them (and hence they are subjective to Hoare himself).

Further, he admonished that language feature design and language design ought to be separate enterprises, the language designer focusing on "consolidation, not innovation" of language features (p. 214). Wirth (1974), after discussing a number of general language design issues, made a similar point: it is the task of the language designer to make decisions where the desiderata are in conflict. Both Hoare and Wirth emphasize that these decisions are primarily based on good engineering. Steele (2006, p. 31) also makes this point: "Good programming-language design requires judgment and compromise"

Steele (1999), in a memorable presentation later published as a journal article, made the point that it is not a good idea to design a large language from scratch, as building it takes too long. Instead, a language should start small, with growth planned for from the beginning.

Unfortunately, there seem to be no contemporary case studies and only a few historical studies of actual language design practices. The available published sources are generally retrospective essays by the designers themselves, typically written for one of the three History of Programming Languages conferences (Wexelblat 1981; Bergin and Gibson 1996; *HOPL III* 2007).

The HOPL conference materials are of limited use, however. As Stern (1979, p. 69) wrote regarding the first HOPL conference:

> "No participant, despite efforts to be objective, can present an unbiased account of his or her own work; no participant can see the whole picture quite as well as an outside observer. Moreover, recollections which are in some cases fifteen to twenty years old are inevitably distorted, whether consciously or unconsciously."

Retrospective essays are useful material but their inherent bias must be taken into an account; a proper historical study is usually preferrable where one exists.

Some peer-reviewed historical studies on the design of high-level programming languages and closely related areas have been published, however, in the (IEEE) Annals of the History of Computing (Marks 1982; Holmevik 1994; Whiting and Pascoe 1994; Giloi 1997; Gray and Smith 2004; Nofre 2010). The Annals has also published some articles on studying history that make comments which are relevant to programming language design (Sammet 1991; Shapiro 1997; Mahoney 2008). There are, of course, other histories of programming languages (e. g. Rosen 1964, 1972; Sammet 1969, 1972; Wegner 1976; Friedman 1992; Knuth and Trabb Pardo 2003; Ryder et al. 2005).

It is beyond the scope of this mapping study to try and generate a coherent theory of past language design practices, but there are some observations that suggest themselves in perusing the materials just cited. First, there is a categorization of languages, suggested by Brooks (1981, p. 683), namely *author languages*

12 By security Hoare meant the lack of undefined semantics, which is more commonly called safety.

versus *committee languages*. He did not define the terms, but the names are suggestive enough; he did, however, note a pattern during the conference: "papers about [committee languages] almost completely concern themselves with process", while "papers about [author languages] have almost completely concerned themselves with technical issues".

I would note that the issue separating author and committee languages from each other is not, in my view, the number of designers or the organizational structure of the development project; instead, it is the development approach: author languages are driven by a single author or a small number of co-authors sharing a technical vision, while committee languages are driven by the need to combine a number of somewhat divergent interests (often represented by stakeholders like expected users or implementors of the language). The development of an author language typically intertwines language definition and implementation, while a committee language is typically clearly defined on its own, with implementation happening elsewhere, and often later.

One author language was the original FORTRAN; Backus (1981, p. 30) described the 1954 vintage design approach as follows:

> "As far as we were aware, we simply made up the language as we went along. We did not regard language design as a difficult problem, merely a simple prelude to the real problem: designing a compiler which could produce efficient programs."

Another obvious author language, until standardization, was C++ (Stroustrup 2014, p. 21 and 23):

> "I invented C++, wrote its early definitions, and produced its first implementation. I chose and formulated the design criteria for C++, designed its major language features[...] In the early years, there was no C++ paper design: design, documentation, and implementation went on simultaneously."

Consider, in contrast, the committee languages Algol (Perlis 1981; Naur 1981; Nofre 2010) and COBOL (Sammet 1981), from the late 1950s, and Haskell (Hudak, Hughes, et al. 2007) from late 1980s. In each case, a committee was formed to draft a new consensus language based on a number of existing languages competing in the same niche: for Algol, the niche was communication of numerical algorithms, for COBOL, the writing of business applications, and for Haskell, lazy functional programming. In each case, the plan was merely take the existing state of the art and combine it into a coherent whole.

Second, it is clear that both author and committee language designs have been mostly driven by technical (and occasionally business) considerations, with implementation concerns, expressive power and the designers' sense of aesthetics being major drivers. Questions of efficacy, that is usefulness to the programmer, are sometimes debated, even fiercely. Many designers (e. g. Cowlishaw 1994; Stroustrup 1994) base their designs on language user feedback, but of the historical treatments of programming languages, only one that I am aware of (Cook 2007) even mentions the possibility of basing design decisions on systematic research of usefulness to programmers.

2.5.2 Programmer behavior

The study of programmers using the empirical techniques of behavioral science is over four decades old. The first somewhat relevant studies were reported in the late 1960s (Sackman et al. 1968; Sackman 1970). The classic text by Weinberg (1971) introduced the topic area and offered quite a bit of analysis but had little to offer in the way of actual empirical results. By the time of the next classic text (Shneiderman 1980), there was already some empirical research that could be discussed (some of it is even relevant to programming language design, and such studies are included in this mapping study). A third book on the topic (Hoc et al. 1990) was published about a decade later; that collection of original articles was able to present detailed psychological theories, backed at least partially by empirical data, on many aspects of programming. A fourth book (Détienne 2002) followed another decade later, and gives a comprehensive synthesis of the field. Traditionally, this field is called the *psychology of programming*, but since I believe there is more than psychology involved – at least cognitive science, sociology (see e. g. Meyerovich and Rabkin 2012), and anthropology are relevant – I use a more inclusive term, (the study of) *programmer behavior*.

At around the time of the Shneiderman (1980) book, three non-systematic surveys were published (Sheil 1981; Arblaster 1982; Hoc 1983). Both Sheil (1981) and Hoc (1983) criticised the extant body of empirical research for serious methodological issues – Sheil (1981) even used rather harsh language in places, for example calling the design of one study "an absurd way to do empirical research" (p. 116) – and the Shneiderman (1980) book for sloppy presentation, which Sheil (1981, p. 116) regarded "the most damaging", as it would lead readers to "reject data that do not support their preconceptions[, which] makes the entire empirical enterprise moot". Very recently, Stefik, Hanenberg, et al. (2014) have, in a systematic secondary study, reviewed studies presented in certain conferences of programmer behavior research for, among other things, research quality, and found them generally poor.

Détienne (2002, p. 1–6) divides the research in the behavior of programming into two phases: "the 1970s" and "the second period". The former is, of course, eponymous, and is rife with serious problems, both methodological and in its basic approach. In this, Détienne echoes the criticisms of Sheil (1981) and Hoc (1983). This contemporary criticism resulted in a paradigm shift that created the second period that lasted at least up to the turn of the millennium, when Détienne was writing. The focus changed, Détienne (2002) recounts, from simple atheoretical "superficial analysis" (p. 6) to the "development of cognitive models of programming".

I will not review the programmer behavior literature in detail here, because that literature is the focus of Chapter 5. There is, however, a line of research not included in this mapping study that I wish to point out: the *cognitive dimensions* model, proposed by Green (1989), is a theoretical framework designed to aid in usability evaluation of notations, like programming languages, and notational systems, like development environments; it might be of use to language designers (see also Blackwell and Green 2003).

A key question is, whether this body of research has influenced actual language designs.[13] As I mentioned earlier, the historical record of language design practice indicates that it has not; this lack of influence was also noted by at least Sheil (1981) and more recently by Hanenberg (2010c) and Markstrum (2010). It is quite possible that this lack of influence is at least partially attributable to the quality issues in existing empirical research. At least the chief language designer of Ada 95, Tucker Taft, reports having "bemoaned the lack of real research into the software engineering advantages or disadvantages of particular design choices" (Ryder et al. 2005, p. 471).

There are two major exceptions to this, and one minor one. First, the Natural Programming project at the Carnegie Mellon University has for nearly two decades applied the research of programmer behavior to programming language and system design (see e. g. Pane and Myers 1996; Pane and Myers 2000; Pane, Myers, and Miller 2002; Myers, Pane, et al. 2004; Pane and Myers 2006; Myers, Ko, et al. 2008). Second, there is the Quorum programming language[14], whose design was influenced by and tested in several published studies of programmer behavior (at least Mayer et al. 2012b; Stefik and Siebert 2013), although it is not clear how much influence the body of research other than that produced by the language authors had on the design, since there does not appear to be a published language design report. Third, the textbook of Klerer (1991) discussed the use of programmer behavior research to inform language design.

2.5.3 Evidence-based?

Let us imagine a practitioner, let us say a family doctor or perhaps a computer programmer. Let us further imagine that they are engaged in the typical task of their profession. For the physician, that would be investigating a particular patient's complaint and coming up with first a diagnosis and then a treatment plan. For the computer programmer, it is the construction of a general solution to a particular class of similar problems by instructing a computer.

Now, let us suppose that they have reached a decision point where they are unsure as to what is the best course forward. The physician may be having doubts whether prescribing a particular medicine is worth its trouble in the case of this particular patient. The programmer may be pondering whether using aspect-oriented programming would be a better choice than object-oriented for solving their particular problem.

Now, in both professions, one might imagine them picking up a reference book, or asking a more experienced coworker. This will frequently appear to solve the problem, in that both the physician and the programmer is likely to form a decision based on that advice.

13 I do not consider here unpublished in-house usability testing of a language design, such as that reported by Cook (2007); it is, of course, desirable, but it does not show the influence of the body of prior research.

14 http://quorumlanguage.com/

There is another approach, one that is commonly advocated under the banner of *evidence-based medicine* (Guyatt 1991; Evidence-Based Medicine Working Group 1992; Straus et al. 2011, and many others), often abbreviated EBM, and *evidence-based software engineering* or EBSE (Kitchenham, Dybå, et al. 2004; Dybå, Kitchenham, et al. 2005).[15] The basic idea in this approach is to put the problematic question to the body of scientific knowledge and to extract an answer that reflects the best scientific evidence available at the time.

Both EBM and EBSE advocate a five-step process for converting a decision point with uncertainty into a decision supported by evidence (Evidence-Based Medicine Working Group 1992, p. 2421; Rosenberg and Donald 1995; Kitchenham, Dybå, et al. 2004, Table 1; Dawes et al. 2005; Dybå, Kitchenham, et al. 2005, p. 59; Straus et al. 2011, p. 3):

1. Ask an answerable question that captures (a part of) your uncertainty in how to proceed.
2. Find the best evidence available that bears on your question.
3. Critically appraise the evidence you found for validity, impact and applicativity.
4. Apply the evidence in solving your practical problem.
5. Evaluate and improve your own performance in evidence-based practice.

Note that this process is intended to be followed by practitioners, not by researchers. Of course, locating and appraising the evidence is too much to ask of a practitioner without support, and thus a number of evidence-based summaries of the literature have been prepared in medicine. In fact, the EBM textbook of Straus et al. (2011) advocates a model called "6S". At the top of the 6S model is a hypothetical patient information system that automatically recognises the answerable questions relevant to the patient's condition and retrieves, for the physician's convenience, the applicable published evidence. At the very bottom are the individual studies, there being too many of them in medicine to be useful to a practicing physician without support from the other Ss. In the middle are abstracts, systematic reviews and (evidence-based) textbooks. Software engineering so far has only accumulated systematic reviews.

This mapping study is based on the conjecture that an *evidence-based programming language design* (EB-PLD) approach might be beneficial. The practitioner in that case would be a programming language designer. Note that this is a different sense of the word than used by Stefik, Siebert, et al. (2011); it also differs from the concept of "evidence-oriented programming languages" envisioned by Stefik, Hanenberg, et al. (2014). However, the detailed study of whether EB-PLD is feasible is beyond the scope of this study.

---

15 Many other disciplines also have adopted an evidence-based paradigm: e. g. management (Rousseau 2006), policing (Sherman 1998), biological conservation (Sutherland et al. 2004), education (Thomas and Pring 2004), and nursing (French 1999).

3 SYSTEMATIC SECONDARY STUDIES

This thesis reports a *secondary* study, one using the published scientific literature (called here the *primary studies*) as its source of data. In this chapter, I will explain the basic concepts and extensively summarize the current methodological guidance, based on the (mainly software engineering) literature. I will close with an examination of the concept of evidence.

3.1 Overview

Within the evidence-based movement there is a distinct preference toward secondary studies being *systematic* (see e. g. Straus et al. 2011). Such secondary studies start with one or more questions that one wants to answer. They perform a systematic search of the literature, to find (as best as one can) all the relevant literature, without bias. They also perform a systematic process of inclusion and exclusion decisions upon the literature found, resulting in a set of publications that (to the best of one's ability) report all relevant studies of sufficient quality. They further perform a systematic process of data extraction and synthesis, yielding answers to the research questions of the secondary study. Most importantly, all the systematic processes used in the study are designed and documented, giving the reader of the study a fair opportunity to evaluate its reliability, and providing an audit trail from the literature to the answers.

There are two main species of systematic secondary studies. *Systematic literature reviews* (also known as *systematic reviews* or *SLR*s) ask specific questions whose answers are immediately relevant to practice; they also involve the synthesis of the results of the studies collected into research-based answers to those practical questions. *Systematic mapping studies*, also called *systematic scoping studies*, ask general questions about the state of the research in a particular (sub)field, often identifying areas lacking research; they usually do not engage in the synthesis of the results reported by individual studies.

The literature is not quite consistent in the use of these two terms; particularly, many studies in software engineering that purport to be systematic reviews are under these definitions more properly classified as systematic mapping studies (e. g. Penzenstandler et al. 2012; García-Borgoñón et al. 2014) as they are intended to guide future research instead of practice (see also Silva, Santos, Soares, França, and Monteiro 2010; Santos and Silva 2013). Similarly, Cruzes and Dybå (2011b) classify some secondary studies self-identifying as systematic literature reviews as scoping studies on the basis that they lack a synthesis of the research results. The three systematic studies on the state of systematic secondary studies in software engineering (Kitchenham, Brereton, Budgen, et al. 2009; Kitchenham, Pretorius, et al. 2010; Silva, Santos, Soares, França, Monteiro, and Maciel 2011) each follow definitions that are essentially the same as mine; Kitchenham, Budgen, et al. (2011, Section 2) discuss a very similar distinction between the two species.

There is a third term that is commonly used in this context: *meta-analysis*. It was originally coined by Glass (1976, p. 3) to mean "the statistical analysis of a large collection of analysis results from individual studies for the purpose of integrating the findings". It is now common to call whole systematic secondary studies meta-analyses, if they use the statistical analysis of primary-study results in their data synthesis (e. g. Brown et al. 2014; Pinsky and Palumbi 2014). It is, however, better to consider meta-analysis merely an umbrella term for certain analysis and synthesis methods (see e. g. O'Rourke 2007), and indeed, many studies label themselves as "systematic review and meta-analysis" (a Google Scholar search of that phrase, limited to titles and the year 2013, conducted on May 2, 2014, reported a hit count of about 3,850).

The research questions I have posed in this thesis are, without doubt, the mapping study kind. They ask about the extent of published research, and are not immediately relevant to practitioners – in this case, language designers. I also do not attempt to synthesize the results of the studies I look at. Hence, this thesis reports a mapping study.

The reason for a secondary study to be conducted systematically is, according to Kitchenham and Charters (2007, p. 3–4), to be "fair and seen to be fair": it "makes it less likely that the results [...] are biased"; it may "provide evidence that [a] phenomenon is robust and transferable"; and it "increases the likelihood of detecting real effects". In balance, one needs to put a lot of effort into making one. In interviews and surveys reported by Zhang and Ali Babar (2013), literature reviewers in software engineering generally agreed with these sentiments.

The claim that systematic secondary studies are particularly trustworthy has been examined in empirical studies to some extent. MacDonell et al. (2010) had two independent teams of experienced researchers perform a SLR on the same topic but designed independently of each other; they found that the two SLRs came to similar conclusions. Kitchenham, Brereton, and Budgen (2012) found, in a case study, that a systematic mapping study can identify publication clusters successfully and perhaps better than a traditional expert review, though their case mapping study did not identify all known relevant studies. Wohlin et al. (2013) conducted two systematic mapping studies that found partially different sets of publications, and came to somewhat different conclusions. They note (p. 2605) that “the reliability of secondary studies cannot and should not be taken for granted”. It seems that the evidence on this topic is mixed.

Petticrew and Roberts (2006, p. 16–17) date the earliest systematic literature review to Nichols (1891). Indeed, while Nichols does not claim to be conducting a systematic review, and while he does not reveal his publication searching and selection methods, the rest of his methodology is very familiar to a modern systematic reviewer, reviewing a number of experiment reports to come to a conclusion on a small number of related, practically relevant, focused questions. Chalmers et al. (2002) refer to an even earlier author as one concerned with the issues that motivate systematic secondary studies. Lind (1757, p. viii) took upon himself

> “to exhibit a full and impartial view of what had hitherto been published on the scurvy; and that in a chronological order, by which the sources of those mistakes might be detected. Indeed, before this subject could be set in a clear and proper light, it was necessary to remove a great deal of rubbish.”

To be sure, his review does not meet the modern standards required for a systematic review, but he did identify one of the key motivators for one.

The modern sense of the term appears to have emerged in the 1970s. Shaikh et al. (1976) conducted a “systematic review” of studies evaluating tonsillectomy, which does not document the literature search but is very strict about evaluating the primary studies under review and synthesizing a result from them. Chalmers et al. (2002) attribute the modern popularity of the phrase to the Foreword written by Archie Cochrane to a 1989 book collecting systematic reviews on obstrectic care. In 1994, an international organization, the Cochrane Collaboration, devoted to the creation and maintenance of a database of systematic reviews in the medical sciences was founded (Bero and Rennie 1995). The ongoing effort to create systematic reviews in software engineering seems to have been initiated a decade ago by Budgen, Boegh, et al. (2003), Kitchenham (2004a), and Kitchenham (2004b).

3.2 Best-practice methodology

In this section, I summarize the state of best-practice methodological guidelines. I focus on software engineering, as it is the discipline that is closest to programming language research with a tradition of systematic secondary studies. I also focus mainly on mapping studies, but discuss SLRs as well to the extent their methodological issues are similar.

Kitchenham and Charters (2007) have published the most recent guidelines for systematic literature reviews in software engineering. They discuss mapping studies only briefly, mostly referring a mapping study researcher to the SLR guidelines. Petersen, Feldt, et al. (2008) augment the guidelines, giving more specific guidance to mapping studies. Kitchenham and Brereton (2013) conducted a systematic review of methodological research regarding systematic secondary studies in software engineering, with a goal of identifying needed changes to the existing SLR guidelines; I will take note of the recommendations they have made, below. Finally, Imtiaz et al. (2013) surveyed published systematic reviews in software engineering for lessons learned about the SLR process itself; the identified lessons are largely similar to the proposals and recommendations I have summarized below and I will not discuss them further.

Many other disciplines also have well-known guidelines for conducting systematic secondary studies. In medicine, the Cochrane Collaboration has published a detailed handbook (Higgins and Green 2011). In the social sciences, there is the textbook by Petticrew and Roberts (2006). However, since Kitchenham and Charters (2007) have explicitly based their guidelines on these sources (although, in the case of the Cochrane handbook, an earlier version), I will not discuss them in detail. Similarly, I will not discuss software engineering SLR literature predating the Kitchenham and Charters (2007) guidelines.

3.2.1 Overall process

Kitchenham and Charters (2007, p. 6) describe 13 distinct phases of a systematic secondary study process, of which 11 they consider mandatory. The major phases are planning, literature search and selection, assessment of the quality of the selected studies, extracting data, and creating a synthesis result from the data. Kitchenham and Brereton (2013, p. 2068) would amend these guidelines to "emphasize the need to keep records of the conduct of the study".

Petersen, Feldt, et al. (2008, p. 2) identify five phases in the conduct of a particular mapping study. The key difference between their process and that of Kitchenham and Charters (2007) is the omission of quality assessment of the included studies. Petersen, Feldt, et al. (2008, p. 7) note that this follows from the different goals of mapping studies versus SLRs: the latter attempt to synthesize the results reported by the individual studies into a coherent collective answer.

Budgen, Turner, et al. (2008) identify largely the same steps for conducting mapping studies. Instead of data extraction and synthesis, they would include the "classification of the available studies" (p. 2). Like Petersen, Feldt, et al. (2008), they consider quality assessment nonessential in a mapping study; Kitchenham, Budgen, et al. (2011) also express a similar opinion.

Some software tools to automate parts (or all) of a systematic secondary study in software engineering have been proposed. A systematic map of them up to 2012 has been published by Marshall and Brereton (2013).

3.2.2 Planning

Planning a review consists of defining research questions and writing a protocol document. Regarding the definition of research questions, Kitchenham and Charters (2007, Section 5.3) give guidelines that are only relevant to SLRs. They recommend, for example, the PICO (population, intervention, comparison, outcome) and PICOC (. . . , context) templates for structuring questions (Petticrew and Roberts 2006; Straus et al. 2011; Higgins and Green 2011), which are appropriate only for questions about relative efficacy (which do not belong in a mapping study). However, Kitchenham and Brereton (2013, p. 2068) consider removing this recommendation from the guidelines appropriate, mostly because it is of limited applicability and value. As to mapping studies specifically, Kitchenham, Budgen, et al. (2011, Table 1 on p. 640) lists "which researchers", "how much activity", and "what type of studies" as typical forms of research question.

The review protocol, per Kitchenham and Charters (2007, Section 5.4), is a document written before the actual systematic secondary study is started, and includes a detailed plan addressing all the phases of the study from identifying a need for it to its dissemination. They also recommend (on p. 14) piloting the protocol before starting the actual study.

3.2.3 Searching

The search for studies, or "identification of research" as Kitchenham and Charters (2007, p. 6) call it, must be properly planned, executed, and documented. Like Petersen, Feldt, et al. (2008, p. 3), they recommend (in Section 6.1) listing words and phrases for each components of the research questions, including synonyms, and using Boolean operators to combine them to form a search term; they recommend searching digital libraries, reference lists of relevant publications, particular journals, particular conference proceedings, and the grey literature (reports that have not been published in well-known academic forums). They also recommend contacting researchers active in the field.

As to electronic databases useful for searching, Kitchenham and Charters (2007, p. 17) specifically mention ACM Digital Library, EI Compendex, Google Scholar, IEEExplore, Inspec, Scopus, ScienceDirect, and SpringerLink. Dieste et al. (2009) also recommend targeting searches on not just reputable general software engineering venues but also on venues specific to the topic area of the secondary study; in some cases, venues of other fields are needed, as well. They thus recommend searching in databases, like Scopus, that cover many venues.

Bailey et al. (2007) and Chen, Ali Babar, et al. (2010) studied the overlap between and contribution of several electronic databases in three and two example systematic secondary studies, respectively, suggesting that using many databases may be necessary; however, their research design make their generalizability beyond the particular example studies doubtful.

Chen, Ali Babar, et al. (2010, p. 2) define three metrics for the performance of a particular "electronic data source" (meaning a particular database, but readily generalizable to any search) in a particular secondary study: the *overall contribution* of a search is the count of relevant publications found by it; the *overlap* between two searches (which should be computed for all unordered pairs of searches, and reported in matrix form) is the number of publications that were found by both, and the *exclusive contribution* of a search is the count of relevant publications found by it and by no other search. Both contribution metrics have, in addition to the absolute count version, a relative version, computed as the ratio of the absolute metric to the total count of relevant studies (with duplicates removed) found in all searches. They suggest that subsequent systematic secondary studies report these metrics, as that would eventually allow a meta-analysis of such studies to generate widely applicable recommendations as to databases to search.

There are two important ratios, borrowed from the field of information retrieval (see e. g. Ceri et al. 2013, p. 7–9), that quantify the performance of a search (see e. g. Petticrew and Roberts 2006, p. 83; Dieste et al. 2009, p. 515; Zhang, Ali Babar, and Tell 2011, p. 627). *Sensitivity*, also called *recall*, quantifies how many of publications that should have been found actually were found. *Specificity*, also called *precision*, quantifies how many of publications that were found were actually publications that should have been found. For the best use of researcher resources, maximizing both ratios is desirable, but as is often the case in multiple-criteria decision problems, there generally is no single optimal search strategy.

More precisely, writing for the moment R for the set of all publications (whether found or not) that are relevant, F for the set of all publications that were found, and $|\cdots|$ for the size of a set, the defining equations are the following:

$$\text{sensitivity} = \frac{|F \cap R|}{|R|} \qquad \text{specificity} = \frac{|F \cap R|}{|F|}$$

Sensitivity is, of course, often impossible to determine accurately, as it requires knowing the extent of the set R, which includes publications that were *not* found. Sometimes, the set R (or a set believed to approximate R well) is called the *gold standard* (e. g. Dieste et al. 2009, p. 516; Zhang, Ali Babar, and Tell 2011, p. 627). However, since both F (the set of all found publications) and $F \cap R$ (the set of all found publications that are relevant) are determined during a systematic secondary study, specificity is usually readily computable.

Dieste et al. (2009) recommend that, when searching for experiments, a researcher should use not just the word "experiment" as a keyword, but also a number of compound terms involving the adjective "experimental", to get good sensitivity and specificity. However, certain other related phrases (like "experimentation" and "empirical study") increase sensitivity modestly while they decrease specificity significantly and are thus not recommended. The keywords should be searched for in article titles and abstracts (not just one of them alone), but widening to other fields is not recommended.

Zhang, Ali Babar, and Tell (2011) propose a disciplined method for defining a query expression for automated searches (see also Zhang, Ali Babar, Bai, et al. 2011). A *quasi-gold standard* (QGS) is, they define, a set of relevant publications published in particular venues during a particular timespan; this set can generally be determined with reasonable use of resources using a manual search and the application of the selection procedure (see next section). A query expression for automated searches can then be elicited by using text mining techniques on the quasi-gold standard, although it is also possible to use ad-hoc query expressions. Then, the ratio of the number of publications in the QGS actually found by the query to the size of the QGS itself, called *quasi-sensitivity*, is computed. The query expression must then be iteratively improved until the quasi-sensitivity meets or exceeds a predetermined threshold. Zhang, Ali Babar, and Tell (2011, p. 629) recommend using a threshold between 70 % and 80 %. Kitchenham and Brereton (2013, p. 2068) consider it appropriate to change the SLR guidelines to recommend this approach.

As already mentioned, Kitchenham and Charters (2007, p. 15) recommend searching in the bibliographies of already identified study reports, a process sometimes called "backward searching" (by e. g. Levy and Ellis 2006). Petticrew and Roberts (2006, p. 98–99) recommend a complementary process that they call "pearl growing" or "forward searching" (the latter also used by e. g. Levy and Ellis 2006), namely "searching for articles which themselves cite a key reference". Both are also an integral part of the search strategy recommended by Webster and Watson (2002) for literature reviews in information systems. The term *snowballing* encompasses both, and appears to be common within the software-engineering systematic secondary study literature (e. g. Budgen, Burn, et al. 2011; Kitchenham, Budgen, et al. 2011; Jalali and Wohlin 2012; Kitchenham and Brereton 2013).

Comparing backward snowballing starting from a known set of papers to database searches, Jalali and Wohlin (2012) found no obvious advantage to either but concluded that they find a slightly different set of papers. A variant of snowballing was also evaluated with encouraging results by Skoglund and Runeson (2009) for software engineering SLRs. Wohlin et al. (2013) in turn conjecture, based on their mapping study reliability study, that snowballing is "more efficient than trying to find optimal search strings" (p. 2605). Further, Kitchenham and Brereton (2013, p. 2068) would amend the SLR guidelines to discuss snowballing more fully.

It is generally expected that all planned searches are exhaustive, that is, everything that is findable by the searches are found and considered for inclusion. Kitchenham and Charters (2007), Budgen, Turner, et al. (2008), Petersen, Feldt, et al. (2008), and Kitchenham, Budgen, et al. (2011) do not discuss this explicitly, but this expectation is clearly implied by them. Petticrew and Roberts (2006, p. 100-101), however, point out that knowing that one has actually achieved finding everything relevant is impossible, and discuss two potential "stopping rules": stopping when key indexes have already been searched and further searching finds very few relevant publications; and stopping when saturation is achieved, that is, when "no further perspectives or schools of thought are added" (quoting Chilcott et al. 2003, p. 7). The proper stopping rule depends, of course, on the particulars of the secondary study.

Kitchenham, Brereton, Turner, et al. (2010) note that a "broad automated search finds more relevant studies than a restricted manual search" and that the results of a systematic secondary study are sensitive to additional studies, except perhaps if low quality publications are excluded. Kitchenham, Brereton, and Budgen (2012) recommend, for "mapping studies of a large body of literature", that "a large and varied set of known studies" be obtained and used in search validation. Publications found via "manual search of important sources" qualify for this set of studies.

3.2.4 Selection

Publications identified by the search efforts must be filtered to select those and only those that are relevant to the secondary study at hand. Kitchenham and Charters (2007, p. 18– 20) recommend that criteria for making this decision, based on the research questions and on practical issues like publication language, be defined in advance and tested. They also recommend an iterative process, first using the title and abstract to exclude clearly irrelevant publications and then looking at the full text of the rest, with possibly a third iteration to enforce a quality threshold. Further, they recommend that, apart from "totally irrelevant" publications, a complete record is kept of exclusion decisions.

Kitchenham and Charters (2007, p. 20) also recommend that either all selection decisions be made by two or more researchers independently or a single researcher working alone retest a random sample of the publications. Petticrew and Roberts (2006, p. 120) make similar recommendations, and also allow a practice where one researcher makes all decisions, with another researcher retesting a random sample. The agreement between researchers (or between the original and the retest) should, say Kitchenham and Charters (2007, p. 20), be evaluated and documented using the Cohen (1960) κ statistic, and any disagreements should be then resolved by discussion. They also recommend a sensitivity analysis in cases where there is uncertainty about the correct decision.

The Cohen (1960) κ (kappa) statistic that Kitchenham and Charters (2007, p. 20) recommend is a measure of *interrater agreement*. It assumes that two people (raters or judges) independently rate a number of items by assigning each of them into one of at least two "independent, mutually exclusive and exhaustive" (Cohen 1960, p. 38) categories. It only works with two independent raters and only if both rate all items. Although Kitchenham and Charters (2007) do not mention it, there is a well-known and commonly used multi-rater κ statistic, the Fleiss (1971) κ. Both κ statistics range from negative values greater than -1 (indicating disagreement beyond mere chance) through 0 (indicating agreement purely by chance) to $+1$ (indicating perfect agreement). There a widely used verbal scale associated with κ statistics, originating from Landis and Koch (1977, p. 165): a negative κ is labeled "poor" agreement, a κ between 0 and 0.2 indicates "slight" agreement, a κ above 0.2 but at most 0.4 indicates "fair" agreement, a κ above 0.4 but at most 0.6 indicates "moderate" agreement, a κ above 0.6 but at most 0.8 indicates "substantial" agreement, and a κ above 0.8 indicates "almost perfect" agreement. However, these verbalizations are "clearly arbitrary" and thus their use is supported by nothing but convention.

Petersen and Ali (2011) have identified a number of strategies researchers in software engineering have used to resolve disagreement about selection decisions. The most common in the secondary studies they identified were a post-selection evaluation of the objectivity of the selection criteria, having another person reviewing the publications in dispute and making the final decision, having researchers discuss the publications in dispute, and letting a publication survive a preliminary round of selection if at least one researcher is uncertain. They recommend that systematic secondary studies report the procedures and decision rules used to make selection decisions in problematic cases.

Malheiros et al. (2007), Tomassetti et al. (2011), Felizardo, Salleh, et al. (2011), and Felizardo, Andery, et al. (2012) propose and validate approaches for using text mining in support of systematic secondary studies, particularly in the selection stage. Kitchenham and Brereton (2013, p. 2068) consider it appropriate for the SLR guidelines to recommend, in the future, that "researchers *consider* the use of textual analysis tools to evaluate the consistency of inclusion/exclusion decisions" (emphasis in the original).

3.2.5 Data extraction and synthesis

With respect to data extraction, Kitchenham and Charters (2007, Section 6.4) recommend defining and piloting "data extraction forms" which direct a researcher to find answers to specific questions selected with the intent that the collected answers can be synthesized into answers to the secondary study research questions. As is the case with exclusion decisions, they recommend that at least two researchers should independently extract data from every included study. In the case of researchers working alone, they allow a retest of a random sample. Turner et al. (2008) reiterate the recommendation of using independent extractions by different researchers (or retesting, in the case of a single researcher), instead of having separate extractor and checker roles.

Care should be taken, Kitchenham and Charters (2007, Section 6.4) recommend, to avoid treating multiple publications reporting the same study as reporting different studies. Kitchenham and Brereton (2013, p. 2068) would amend the guidelines to "mention the need to report how duplicate studies are handled."

Kitchenham and Charters (2007, Section 6.5) recommend tabulating the extracted data, highlighting similarities and differences between the studies. Beyond this, their recommendations make sense only in the context of synthesizing outcomes, which mapping studies generally (and this mapping study specifically) do not do.

While outcome synthesis is not particularly relevant to a mapping study, it must be noted that even systematic literature reviews in software engineering have not, at least until recently, properly considered the problem of such synthesis, according to Cruzes and Dybå (2011b).

Petersen, Feldt, et al. (2008, p. 3–5), for their part, describe a two-stage process of creating a systematic map, in which data extraction and synthesis are intertwined. They first had researchers identify, for each paper included, keywords that "reflect the contribution of the paper" in the paper's abstract (and in some cases, its introduction and conclusion sections). These keywords were the interpretation of the researchers themselves, and need not be the same as any keywords chosen by the authors of the paper under study. The results were then combined, yielding a classification scheme for papers. Then, each paper was classified according to the scheme. The resulting systematic map consisted of the various frequencies of publications in each category. For visualizing the map, they recommend a *bubble plot*, a scatterplot in which each data point is drawn as a circle, the area of which being proportional to the magnitude of the data point – in this case, the magnitude being the frequency of publications.

Cruzes and Dybå (2011a) introduce a method for synthesizing outcomes of qualitative primary studies in SLRs, although the method makes sense also in the mapping study context. This *thematic synthesis* method proceeds by identifying relevant passages in the primary studies, then assigning codes[1], then creating themes out of the codes, and finally generating a thematic model of the primary studies.

Felizardo, Riaz, et al. (2011) recommend, based on a controlled experiment, presenting synthesis results using edge–node graph drawings. Their experimental setup tested this recommendation on a quintessentially mapping-study data set, the relationship between articles and publication years, and thus their recommendation, although phrased as applying to SLRs, is readily applicable to mapping studies.

Cruzes, Mendonça, et al. (2007) and Felizardo, Nakagawa, et al. (2010) suggest that text-mining tools be used in the data extraction phase of systematic secondary studies. The technique advanced by Felizardo et. al is particularly designed for generating a systematic map. Nieminen et al. (2013) introduce a knowledge discovery approach to creating a nonsystematic map of a research field, which probably can be adapted to function as a part of a systematic secondary study process.

3.2.6 Reporting

Kitchenham and Charters (2007, p. 40) recommend that all systematic secondary studies be reported both as a journal or a conference paper and as a technical report or a thesis. They also recommend that the journal or conference paper, having usually a length limit, refer to the technical report or thesis for details omitted in the paper. They further recommend that some sort of peer review be performed on all to-be-published systematic secondary study reports, including technical reports that are not typically subject to it, if they are published on the World Wide Web.

Kitchenham, Brereton, Li, et al. (2011) recommend, based on a case study involving two independent SLRs on the same topic, that systematic secondary studies be reported in detail, including documenting search strings and the selection criteria, so that there is a chance for the study to be repeatable.

Kitchenham, Brereton, and Budgen (2012) recommend that mapping study reports cite all publications related to included studies, not just the most recent

---

[1] *Codes*, in qualitative research, are labels given to passages of text, describing the content of those passages for an analytic purpose; the process of assigning codes is called *coding* (see e. g. Schwandt 2007, entry for "coding" on p. 33–34). Despite the similar terminology, this has nothing to do with writing computer programs.

or most complete, even though analysis and synthesis must of course merge the duplicates.

3.2.7 Concluding remarks

I have now summarized the current recommended practice for systematic secondary studies primarily in software engineering. These recommendations influenced the design of this mapping study. To some extent, this mapping study does not comply with all of these recommendations, mostly because this study was designed before they were published, but also to some extent due to the fact that I misjudged the relevance of some of them (particularly Zhang, Ali Babar, and Tell 2011; Zhang, Ali Babar, Bai, et al. 2011, describing the quasi-gold standard method of iterative literature searching) at the time of their publication.

3.3 On evidence

A systematic secondary study is most often about locating and summarizing evidence. What evidence is seems obvious at first, but, as Vesa Lappalainen demonstrated to me in personal communication, reveals significant hidden uncertainty and depth upon closer inspection. This mapping study explicitly looks for evidence, and therefore a clear definition had to be developed to guide literature searching. A full development of the issue is beyond the scope of this mapping study, but the key ideas and arguments are outlined below.

3.3.1 Research methods

In a systematic secondary study, the implicit context is that one is looking for research evidence, that is, scientific or scholarly studies duly reported that bear on the subject at hand (collectively called *primary studies*). In the behavioral sciences, which are the most relevant for this study, a number of research methods have become standard; they are conventionally classified into the *quantitative* and the *qualitative*.[2]

There are a number of qualitative methods, including *case study* (Yin 2009; Runeson et al. 2012), *content analysis* and *thematic analysis* (see e. g. Vaismoradi et al. 2013), *grounded theory* (Glaser and Strauss 1967), *ethnography* (see e. g. Crabtree et al. 2009; Morrison et al. 2010), and *action research* (see e. g. Avison et al. 1999). Common to all of them is a focus on the particulars of a specific situation and attempting to achieve a deep understanding of it, and sometimes a beneficial change in it, instead of generalization into putatively universal laws. Commonly, the situation is looked at from the point of view of the participants instead of

[2] Vessey, Ramesh, et al. (2005) developed a classification of, among other things, research methods in computing, partially based on Alavi and Carlson (1992). I find these classifications not very useful, as they do not define their terms very clearly (see Section 6.1).

the point of view of an outside observer. It should be noted that the mere use of qualitative data (such as interviews) does not make a study qualitative in nature.

In quantitative research the goal is typically to estimate the effect of one or more *treatments* (the choice of treatment, including perhaps their absence, form the *conditions*, also referred to as the values of the *independent variables*) on one or more quantities of interest, the *dependent variables*, with the goal of testing theories consisting of (qualified) universal laws and asserting a causal connection between the independent and dependent variables. The methods are broadly categorized (see e. g. Whitley et al. 2013, p. 36–45) into the *experimental* approach, in which the researchers control to various degrees the circumstances and conduct of the research, and the *correlational*, in which the researchers observe real-life phenomena without exerting control over them.

Experimental studies have, according to Whitley et al. (2013, p. 242), three defining characteristics: "manipulation of the independent variable", "holding all other variables in the research situation constant", and "ensuring that participants in the experimental and control conditions have equivalent personal characteristics and are equivalent with respect to the dependent variable before they take part in the experiment". They can be *between-subjects* designs, in which the various treatments and perhaps their absence are assigned to different people (forming *experimental groups* and a *control group*, the latter being given no treatment or a control treatment), and the result is obtained by examining the difference in the dependent variable values between the groups (Whitley et al. 2013, p. 252–255). Alternatively, they can be *within-subjects* (or *repeated measures*) designs, in which each participant is sequentially subjected to each of the experimental treatments and the control treatment in turn, and the result is obtained by considering the change in the dependent variables; within-subjects designs can be *counterbalanced*, in which the participants are divided into several groups, each getting the treatments in a different sequence (Whitley et al. 2013, p. 255–259). *Factorial designs* can be used to measure the effect of several independent variables in the same experiment (Whitley et al. 2013, p. 264–255), in which case the experiment may be within subjects for some variables and between subjects for others.

Campbell and Stanley (1963) further classified experimental study designs into three categories: *pre-experimental* designs, *true experimental* designs, and *quasi-experimental* designs. True experiments they defined to be experiments following all contemporary recommendations on experiment design, particularly the use of a control group for which the treatment is absent, and assignment of participants to the groups by a random process. Pre-experimental designs predate the establishment of these standards and generally fall short of them, though they can be successful in limited circumstances. Quasi-experiments are experimental studies that lack one or more of the requirements imposed on true experiments due to circumstances of the experiment that preclude their employment (one supposes that if a design fails to meet the criteria for some reason attributable not to the circumstances but to the researchers, the study would be classified as pre-experimental and not quasi-experimental). They include within-subjects designs (even counterbalanced ones) in the quasi-experimental category.

In this mapping study, I will assign experiments into three categories. The most broad category is that of *experiments*: studies in which the researchers attempt to influence one or more independent variables in order to cause changes in one or more dependent variables. The next category is that of *controlled experiments*: experiments in which the experimental subjects (which, if human, are called *participants*), are assigned into groups based on which treatment (or their absence) they are subjected to and in which sequence. I further require that in controlled experiments the groups cover all treatments (including their absence, if no control treatment is used) and all the possible sequences in which they are administered. Thus, I categorize a within-subjects experiment as controlled only if it is completely counterbalanced. The third category is that of *randomized controlled experiments* (also often called *randomized controlled trials*): controlled experiments in which subjects are assigned into groups by a random process. Note that there are studies that I categorize as experiments that Whitley et al. (2013) would not; further, while I believe all the Campbell and Stanley (1963) true experiments qualify as randomized controlled experiments, not all randomized controlled experiments are true experiments.

3.3.2 Hierarchies of evidence

In Evidence-Based Medicine, the concept of evidence is often simplified into a *hierarchy of evidence*. For example, the Oxford Centre for Evidence Based Medicine (Howick et al. 2011) allocates, for assessing treatment benefits, the following levels: Level 1 consists of systematic reviews of randomized controlled trials, Level 2 of individual randomized controlled trials or correlational studies that demonstrate a "dramatic effect", Level 3 of individual non-randomized controlled trials, Level 4 of certain other types of studies, and Level 5 of reasoning from theoretical knowledge.

This simplification of the concept of evidence should not be confused with the real thing, as that leads to absurd results, both serious (Atwood 2008; Hammerstrøm and Bjørndal 2011) and humorous (Smith and Pell 2003). Both problems stem from an overly rigid interpretation of the evidence hierarchy, trusting randomized controlled trials over all other evidence, however convincing the latter are on their own terms. After all, there is an inherent weakness in all statistical studies, namely the possibility that a positive result is actually false (even if the study is methodologically flawless), which is significantly magnified when the prior probability of an effect is small and when the true effect, if present, is small (for a recent well-known case and its aftermath, see Bem 2011; Wagenmakers et al. 2011; Francis 2012; Fiedler and Krueger 2013; more generally, see Ioannidis 2005, 2008; see also Every-Palmer and Howick 2014)

Further, as Cartwright and Stegenga (2011) – writing in the context of evidence-based policy – point out, a traditional hierarchy of evidence with randomized controlled experiments at the top and theoretical inferences at the bottom is only one aspect of evidence that is relevant to its potential user: equally important are its relevance and its evaluation in the specific context of proposed use. There are relevant empirical questions, for which evidence is desirable, for which the traditional hierarchy is the wrong approach, according to Cartwright and Stegenga (2011), for example the causal structure of the proposed context of use. Dybå, Sjøberg, et al. (2012) also stress the importance of context in empirical software engineering and how that context is practically impossible to control for in an experiment, reducing the usefulness of controlled experiments in that field; there does not seem to be any reason to suspect the programming language field is spared from this.

3.3.3 On the epistemology of evidence

In this mapping study, I approach evidence without a preconceived hierarchy in mind (I do, however, use an evidence hierarchy in the analysis and synthesis of the included studies). The main criterion I use is whether a study provides scientific empirical evidence on a relevant question. This approach, however, requires me to confront the question of what evidence actually is. This is a question of epistemology; although a proper study of the relevant questions is beyond the scope of this study, I will sketch the main argument.

First, I must dismiss a number of historical epistemological stances. First is the idea that a series of successful empirical tests of a theory confirms a theory; the second is the idea, due to Popper (1980), that the only thing we can say of a theory is that it has or has not been falsified. The untenability of the former is well known (see e. g. Russell 1983, p. 35). The idea of falsification fails as well, for two separate reasons: the status of not-yet-falsified is absolutely useless when one must choose among several such theories, and as Quine (1951) noted, it is always possible to react to an empirical refutation of a theory by tweaking the theory (and in many cases this is even the right choice). These arguments are introductory-textbook material in the philosophy of science (see e. g. Bird 1998; Godfrey-Smith 2003).

In the social sciences, there are two major epistemological traditions of research. Each generally (though not universally) dismisses the other, sometimes with strong harsh words. One tradition, self-labelled as *antipositivism*, has given the other the (often pejoratively intended) label *positivism* (this label is, however, historically inaccurate, see Mackenzie 2011) and regards it as a decades earlier thoroughly discredited research paradigm (for a recent antipositivist formulation, see St. Pierre 2012). The antipositivists themselves divide into several sub-camps each having a label, proudly worn by its members, such as *critical theory*, *feminism*, and *constructivism* (for an overview, see Guba and Lincoln 1994).

On the other side of the divide, the researchers who are given the positivist label do not typically use that (or any other) label of themselves; they merely see their approach as good scientific practice and regard the antipositivist approaches as unscientific or worse (for a recent strongly worded formulation, see Colquhoun 2011, p. 336–339), and often just ignore them.

The reason for these divisions is a fundamental difference in ontological, epistemological, and axiological views which results in different and perhaps even incompatible methodology and standards of good research (for a summary written by antipositivists, see Lincoln et al. 2011). The antipositivists generally avoid quantitative methods, while the other tradition embraces them; hence, these two traditions are often (somewhat incorrectly) called the *qualitative* and the *quantitative* paradigms, respectively.

In this study, I do not wish to take a firm stand for or against either approach; however, the very fact that I am working within an evidence-based paradigm (as well as my methodology here generally) does bias this study against the antipositivists somewhat (see e. g. Suri 2013). Instead, I have attempted to formulate an epistemological position that is reasonably agnostic on this issue.

There are, in recent philosophy of science, two main approaches to epistemology. One is *inference to the best explanation* (see e. g. Lipton 2004), and the other is *Bayesianism* (see e. g. Howson and Urbach 2006; Jeffrey 2004). Some authors (such as Godfrey-Smith 2003) are of the opinion that they are incompatible, but like Lipton (2004), I believe them to be compatible. In any case, for the purposes of this mapping study, the Bayesian approach is more instructive.

The central idea of Bayesian epistemology is that the proper way to assess a claim is to assign it a probability. A probability assignment based on the totality of current knowledge about the claim is called a *prior probability* or just a *prior*; when a new piece of knowledge is added, the prior is transformed into a new probability, the *posterior probability* or just the *posterior*. When another new piece of knowledge is about to be added, the old posterior becomes the new prior, and the new piece creates a new posterior.

Some Bayesians (e. g. Jeffrey 2004) posit that Bayesianism is about the ideal rational person, the *Bayesian agent*, defined as having the following characteristics: if it were to place bets based on its beliefs, it would not be vulnerable to a Dutch book – a set of bets which is certain to result in a net loss, such as betting against the ordinary mathematical statement $1 + 1 = 2$, but usually more complex – and it reacts to observations by adopting the posterior probability suggested by a Bayesian analysis as its new prior. Others (e. g. Howson and Urbach 2006) regard the Bayesian theory of probability as a *logic of induction*, on a par with the more familiar logics of deduction (such as elementary first-order logic); it does not define a rational being, merely what it means to be rational.

From this point of view, the meaning of "evidence" becomes plain. First, evidence is an observation, something external to the observer that the observer becomes aware of. Second, evidence requires interpretation. Third, evidence never exists in isolation, rather all evidence is evidence about some proposition. In sum, evidence about a proposition is an observation that a rational person interprets as changing their confidence in that proposition. In other words:

> **Definition 4.** *Evidence* comprises reported observations about the contingent aspects of the world. Evidence is *about* a claim if it has the potential to affect a rational person's confidence in the claim. Evidence is *scientific* if it has been honestly, systematically and deliberately collected for a research purpose. Plain assertions, descriptions of functionality, anecdotes, expert opinions, personal experience reports by the researchers, and formal proofs are not scientific empirical evidence.

This may be just an artifact of Bayesianism but I adopt it here: evidence is inherently empirical. A logically valid or contradictory proposition has, for all Bayesian agents, probability of one or zero, respectively, which no observation can possibly change within the Bayesian logic. Only contingent propositions can have evidence.

Definition 5. A proposition is *contingent* if it there are possible worlds where it is true and possible worlds where it is false. In other words, a contingent proposition is not a logical tautology nor a logical contradiction; a Bayesian agent would know its truth value a priori.

Note that this definition sidesteps the notorious problem of old evidence often attributed to Bayesianism: how can an observation known to a Bayesian agent be first dismissed as irrelevant but later be recognized as evidence, which is something that happens in actuality? My definition does not require a person to be a Bayesian agent, it merely speaks of a hypothetical "rational person", which is a Bayesian agent. When one recognizes an old observation as evidence, one is essentially realizing that a Bayesian agent would change its confidence in the proposition at hand upon observing it.

I have used these definitions in the mapping study as an aid to decide when a study provides (empirical) evidence and when it does not. The following chapter details the actual process, both for making those decisions and for other aspects of this study.

4 THE MAPPING PROCESS

This thesis reports a systematic mapping study; this chapter explains the mapping process used. A high-level view to the process is shown in Figure 1.

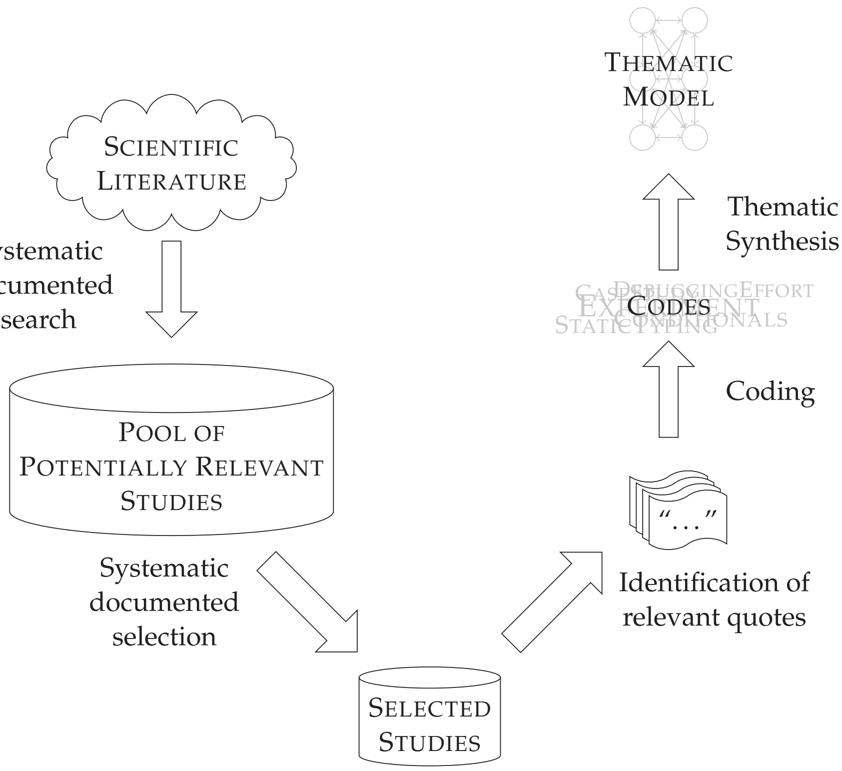

FIGURE 1 A high-level representation of the mapping process. This diagram omits many details.

First, I wrote a protocol document that described the planned process before any actual work commenced. As the study progressed, I revised the protocol several times. My supervisors reviewed the original protocol document and all revisions before I started following them. I did not request an external review of the protocol. The protocol and all its revisions are available from me by request.

Throughout the study, I maintained a record on the process, including all the intermediate data generated during the process. The records form a text-based, both machine- and human-readable database under version control. Appendix 1 details the database and the tools I used.

I searched for candidate studies by manual and automatic search of various venues and databases and in several iterations, as discussed in Section 4.1. I then proceeded to decide which of the potential studies should be included in three phases: Phase I was a preliminary selection phase, in which only the most obvious cases were excluded, and the rest were retained for more careful checking in Phase II; I considered only on-line metadata in these two phases. In Phase III, I obtained the full text of all studies that survived the previous phases, and made my provisionally final decisions. I also conducted a single iteration of snowball search on studies that had provisionally been selected for inclusion; this yielded additional candidate studies that I then subjected to selection Phases I through III. After a selection evaluation exercise, the decisions were finalized. The selection process is discussed in Section 4.2.

After final selection decisions, I conducted a four-stage thematic synthesis process, as discussed in Section 4.3. I first read all studies selected for inclusion. Then, I extracted from the studies direct quotes that appeared relevant to the research questions. I then developed a coding scheme, and applied it to these quotes. I finally created a thematic model of the included studies.

4.1 Searching for candidate studies

The search for candidate studies consisted of three phases: manual search, automatic search, and snowball search. This process is summarized in Table 1.[1]

The first iteration of manual and automatic searches took place from December 2010 to September 2011. A second iteration of manual and automatic searches was conducted in December 2012 and January 2013, to update the set of candidate studies to include studies published up to 2012. The single iteration of snowball search took place between February and April 2013.

4.1.1 Manual search

I conducted a manual search (summarized in Table 2) of the following journals and conference proceedings series, which I believed to be the most relevant venues in programming language research and in empirical studies of software engineering and of programmers:

[1] The table shows article counts as of this writing. They do not necessarily reflect the situation at the indicated end dates, as subsequent developments have revealed duplicates in the article database, which have been merged as discovered.

TABLE 1 Summary of selection process. This table does not show selection validation and the resulting changes in inclusion/exclusion decisions, nor does it show exclusions made *post hoc* during data extraction.

	From	To	Passed	Excluded
First iteration – initial searches				
– Phase I	Dec. 9, 2010	Sep. 16, 2011	1515	
– Phase II	Sep. 17, 2011	Nov. 24, 2011	1045	470
– Phase III	Nov. 24, 2011	Apr. 30, 2013	92	953
Second iteration – search update up to 2012				
– Phase I	Dec. 20, 2012	Jan. 10, 2013	248	
– Phase II	Jan. 9, 2013	Jan. 23, 2013	151	97
– Phase III	Jan. 24, 2013	Feb. 18, 2013	26	125
Third iteration – first round of snowballing				
– Phase I	Feb. 15, 2013	Mar. 12, 2013	293	
– Phase II	Mar. 13, 2013	Mar. 19, 2013	223	70
– Phase III	Mar. 19, 2013	Apr. 30, 2013	68	155

– ACM Transactions on Programming Languages and Systems (TOPLAS)
– ACM Letters on Programming Languages and Systems (LOPLAS)
– Communications of the ACM (CACM, up to 1990)
– Empirical software engineering (ESE)
– European Conference on Object-Oriented Programming (ECOOP)
– ACM SIGPLAN International Conference on Object-Oriented Systems, Languages, and Applications (OOPSLA)
– ACM International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH)
– ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages (POPL)
– ACM/IEEE International Symposium on Empirical Software Engineering (ISESE)
– ACM/IEEE International Symposium Empirical Software Engineering and Measurement (ESEM)
– Symposium of the Psychology of Programming Interest Group (PPIG)
– International Journal of Man–Machine Studies (IJMMS)
– International Journal of Human–Computer Studies (IJHCS)

Around the year 1990, the CACM was repositioned as a magazine targeting the ACM's membership rather than the research community (Denning 1989). Thus, the CACM does not seem crucial enough a forum after 1990 to warrant manual searching. The IJMMS and the IJHCS were added to this list after the first iteration of searches had uncovered a number of articles in the IJMMS, making it likely that it, and its successor the IJHCS, would contain more relevant articles.

TABLE 2 Summary of manual search

(a) Journals

Journal	Vols.	Years	Source	Date of search	Yield*
ESE	1–17	1997-2012	Springer	Dec. 10, 2010, Jan. 4, 2013	9
CACM	1–33	1958–1990	ACM	Dec. 13–21, 2010, Jan. 17–19, 2011	280
TOPLAS	1–34	1979–2012	ACM	Dec. 17–20, 2010, Jan. 7–10, 2013	182
LOPLAS	1–2	1992–1993	ACM	Dec. 21, 2010	7
IJMMS	1–39	1969–1993	SD	Dec. 20–21, 2012	82
IJHCS	40–70	1993–2012	SD	Dec. 21, 2012, Jan. 4, 2013	27

(b) Conference proceedings

Proc. of	Years	Source	Date of search	Yield*
PPIG	1989-2012	PPIG†	Dec. 9, 2010, Jan. 4, 2013	63
ISESE	2002–2006	ACM, IEEE	Dec. 10, 2010	1
ESEM	2007–2012	ACM, IEEE††	Dec. 10, 2010, Jan. 4, 2013	8
OOPSLA & SPLASH	1986–2012	ACM	Jan. 19–28, 2011, Jan. 7, 2013	207
ECOOP	1987–2012	Springer†††	Jan. 28, Feb. 1–7, Jun. 1–17, Aug. 4–19, 2011, Jan. 4, 2013	286
POPL	1973–2012	ACM	Aug. 19–22, Sep. 1, 2011, Jan. 4, 2013	219

Source abbreviations: ACM = ACM Digital Library, IEEE = IEEE Xplore, Springer = SpringerLink, SD = ScienceDirect, PPIG = http://ppig.org/workshops/
* Yield refers to the number of candidate publications recorded.
† Except for the year 2012, where http://ppig2012.eventbrite.com/ (accessed on January 4, 2013) was used.
†† Except for the year 2012, where http://esem.cs.lth.se/esem2012/esem/program.shtml (accessed on January 4, 2013) was used.
††† Except for the year 1989, where http://www.ifs.uni-linz.ac.at/~ecoop/cd/tocs/tec89.htm (accessed on February 7, 2011) was used.

4.1.2 Automatic search

While manual searching of specific publication venues can be very reliable so far as the venues themselves are concerned, it completely ignores any publications in other venues. To achieve better coverage of the full field of relevant publications, I performed additional keyword-based searches of literature and citation databases, summarized in Table 3.

I developed the search phrase used in this study from the following reformulation of the study goal:

> to find empirical studies regarding the impact of design decisions on programming language's influence on the programming process.

I considered each of the key phrases in the reformulation separately, to form a set of key phrases:

empirical study: This mapping study is limited to empirical studies. However, the exact phrase "empirical study" is not likely to appear in all relevant papers. The word "empirical" alone is more likely, but it is also likely to appear in many irrelevant papers. Requiring that the word appears in the article title narrows the set of matches quite a lot and is likely to drop many relevant studies as well. This can be mitigated by listing likely empirical research methods, selected from among those listed by Glass et al. (2002) and Ramesh et al. (2004): "experiment", "action research", "case study", "ethnography", "field study", "grounded theory", "hermeneutics", "literature review", "meta-analysis", and "phenomenology". Adding likely variants, I ended up with the following disjunctive compound:

$$\begin{gathered} M := \\ \text{empirical} \lor \text{experiment} \lor \text{experimental} \lor \text{action research} \lor \\ \text{case study} \lor \text{ethnography} \lor \text{ethnographical} \lor \text{field study} \lor \\ \text{grounded theory} \lor \text{hermeneutics} \lor \text{hermeneutical} \lor \text{literature review} \lor \\ \text{meta-analysis} \lor \text{meta-analytical} \lor \text{phenomenological} \lor \text{phenomenology} \end{gathered}$$

impact of design decisions: I dropped this phrase, because there did not appear to be any variants of it that are found in the relevant studies but not in lots of other studies.

programming language: Any relevant study will contain this term; there are many irrelevant studies that won't. Thus, I retained it unchanged.

influence on the programming process: I dropped this phrase, because there do not appear to be any variants of it that are found in the relevant studies but not in lots of other studies.

Thus, the search phrase used is simply

$$\text{programming language} \land M$$

(with M restricted to article titles) adapted to the query language of each search engine at hand.

I performed this search in the following databases:

TABLE 3 Summary of automatic search

Engine	Search expression	Years	Date	Hits*	Yield†
Google Scholar	`"programming language" (intitle:hermeneutics OR intitle:hermeneutical OR intitle:"literature review" OR intitle:"meta-analysis" OR intitle:"meta-analytical" OR intitle:phenomenological OR intitle:phenomenology)`	all	Sep. 5, 2011	161	9
ScienceDirect	`"programming language" AND title(empirical OR experiment OR experimental OR "action research" OR "case study" OR ethnography OR ethnographical OR "field study" OR "grounded theory" OR hermeneutics OR hermeneutical OR "literature review" OR "meta-analysis" OR phenomenological OR phenomenology)`	all	Sep. 5, 2011	870	45
IEEE Xplore	`"programming language" AND ( "Document Title":empirical OR "Document Title":experiment OR "Document Title":"action research" OR "Document Title":"case study" OR "Document Title":ethnography OR "Document Title":ethnographical OR "Document Title":"field study")`	all	Sep. 6, 2011	862	57
Google Scholar	`"programming language" (intitle:"empirical" OR intitle:"experiment")`	all	Sep. 7, 2011	2050‡	99
Google Scholar	`"programming language" (intitle:"empirical" OR intitle:"experiment")`	up to 2000	Sep. 12, 2011	659	83
Google Scholar	`"programming language" (intitle:"empirical" OR intitle:"experiment")`	2001–2005	Sep. 12, 2011	418	26
Google Scholar	`"programming language" (intitle:"empirical" OR intitle:"experiment")`	2006 onward	Sep. 13, 2011	667	39
Google Scholar	`"programming language" intitle:"experimental"`	up to 2000	Sep. 14, 2011	494	45
Google Scholar	`"programming language" intitle:"experimental"`	2001 onward	Sep. 14, 2011	676	17
Google Scholar	`"programming language" intitle:"action research"`	all	Sep. 15, 2011	13	0
Google Scholar	`"programming language" intitle:"case study"`	up to 2002	Sep. 15, 2011	932	83
Google Scholar	`"programming language" intitle:"case study"`	2003–2007	Sep. 16, 2011	594	11
Google Scholar	`"programming language" intitle:"case study"`	2008 onward	Sep. 16, 2011	510	14
Google Scholar	`"programming language" (intitle:ethnography OR intitle:ethnographical OR intitle:"field study" OR intitle:"grounded theory")`	all	Sep. 16, 2011	59	6
Web of Science	`TI="programming language" AND TI=(empirical OR experiment OR experimental OR "action research" OR "case study" OR ethnography OR ethnographical OR "field study" OR "grounded theory" OR hermeneutics OR hermeneutical OR "literature review" OR "meta-analysis" OR "meta-analytical" OR phenomenological OR phenomenology)`	all	Sep. 16, 2011	19	9
IEEE Xplore	`"programming language" AND ("Document Title":"grounded theory" OR "Document Title":hermeneutics OR "Document Title":hermeneutical OR "Document Title":"literature review" OR "Document Title":"meta-analysis" OR "Document Title":"meta-analytical" OR "Document Title":phenomenological OR "Document Title":phenomenology)`	all	Sep. 16, 2011	3	0
Google Scholar	`"programming language" (intitle:hermeneutics OR intitle:hermeneutical OR intitle:"literature review" OR intitle:"meta-analysis" OR intitle:"meta-analytical" OR intitle:phenomenological OR intitle:phenomenology)`	2011–2012	Jan. 7, 2013	61	2
Google Scholar	`"programming language" (intitle:"empirical" OR intitle:"experiment")`	2011–2012	Jan. 7, 2013	382	28
Google Scholar	`"programming language" intitle:"experimental"`	2011–2012	Jan. 8, 2013	254	2
Google Scholar	`"programming language" (intitle:"action research" OR intitle:"case study" OR intitle:ethnography OR intitle:ethnographical OR intitle:"field study" OR intitle:"grounded theory")`	2011–2012	Jan. 8, 2013	525	15
IEEE Xplore	`"programming language" AND ( "Document Title":empirical OR "Document Title":experiment OR "Document Title":"action research" OR "Document Title":"case study" OR "Document Title":ethnography OR "Document Title":ethnographical OR "Document Title":"field study")`	2011–2012	Jan. 9, 2013	129	11
IEEE Xplore	`"programming language" AND ("Document Title":"grounded theory" OR "Document Title":hermeneutics OR "Document Title":hermeneutical OR "Document Title":"literature review" OR "Document Title":"meta-analysis" OR "Document Title":"meta-analytical" OR "Document Title":phenomenological OR "Document Title":phenomenology)`	2011–2012	Jan. 9, 2013	1	1
ScienceDirect	`"programming language" AND title(empirical OR experiment OR experimental OR "action research" OR "case study" OR ethnography OR ethnographical OR "field study" OR "grounded theory" OR hermeneutics OR hermeneutical OR "literature review" OR "meta-analysis" OR phenomenological OR phenomenology)`	2011–2012	Jan. 9, 2013	152	5
Web of Science	`TI="programming language" AND TI=(empirical OR experiment OR experimental OR "action research" OR "case study" OR ethnography OR ethnographical OR "field study" OR "grounded theory" OR hermeneutics OR hermeneutical OR "literature review" OR "meta-analysis" OR "meta-analytical" OR phenomenological OR phenomenology)`	2011–2012	Jan. 9, 2013	1	1

* Hits refers to the number of search results obtained, as reported by the search engine.
† Yield refers to the number of candidate publications recorded (may include some of the same candidates as other searches).
‡ See discussion in the text (p. 95).

- Google Scholar
- IEEE Xplore
- ISI Web of Science
- ScienceDirect

The following databases were considered and rejected:

- ACM Digital Library, because of an insufficient search language
- EI Compendex, because I was not familiar with it
- SpringerLink, because of an insufficient search language
- SCOPUS, because I did not have access to it at the time

After the protocol-indicated searches had been completed and selection had been commenced, at the suggestion of a colleague, on 29 September 2011, I reassessed the viability of ACM Digital Library and SpringerLink for direct keyword searching. I made the following observations:

- ACM Digital Library provides two kinds of searches. A simple search box is provided (apparently) with no guidance as to syntax. There is a link to advanced search, which allows a multitude of structured queries but apparently not what this study needs: a phrase search in all fields conjuncted with a word or phrase in the title field. However, a Google search for '"acm digital library" search help' reveals a "Search Help" page[2]. It reveals that phrases can be indicated by using double quotes and that space separation is interpreted as conjunction (ACM s.d.[a]). However, even though there is an indication that searching fields is supported (ACM s.d.[b]), trial searches indicate that this is not the case: for example, the search '"programming language" title:experiment' retrieves no matches. Also, the documentation appears to be generic help for a search engine that ACM Digital Library presumably uses but has not been reviewed and customized to match the actual situation in the Digital library (see the note at the end of ACM s.d.[b]).
- SpringerLink also provides two kinds of searches. SpringerLink (s.d.) indicates that the simple search box supports Boolean searches, but there appears to be no way to restrict particular components of the search to a field such as title. Advanced search allows structured searching but even it does not allow searching on both all fields and a specific field at the same time.

The ability to restrict some but not all of the keywords to the title field is an important part of the search strategy and allows controlling the size of the result set. Accordingly, both search engines are considered unfit for this study. As earlier indicated, this is mitigated by the fact that Google Scholar indexes both databases. All Google Scholar searches found 49 articles bearing a 10.1007 (Springer) DOI (of a total of 330 articles with such DOIs found in all searches), and 80 articles bearing a 10.1145 (ACM) DOI (967 total); but it should be noted that these numbers are before selection and thus the totals contain quite a bit of irrelevant hits.

---

[2] http://dl.acm.org/search_help.cfm

4.1.3 Snowball search

After manual and automatic searches, I made selection decisions regarding all of the publications located, as described in the next section. I then subjected each publication that I had selected for inclusion to a snowball search:

- I scanned by eyeball the references list of each such article.
- I also searched for the article in ACM Digital Library, Google Scholar and ISI Web of Science, and scanned by eyeball the lists of citing articles that each database returned.

The publications uncovered by this snowball search were then submitted to a new iteration of selection (see the next section).

Snowball searching occurred between February 15 and March 12, 2013. Due to time pressure, I conducted only one round of snowballing; that is, references uncovered during snowballing that survived selection were not submitted to snowball searching.

4.1.4 Validation

During protocol development, I identified four articles that should be found by the searches: Hanenberg (2010a), Hanenberg (2010b), Malayeri and Aldrich (2009), and Prechelt and Tichy (1998). All four were found. The rest of this validation is *post hoc*, not considered in the protocol.

Table 4 shows the overall and exclusive contribution of each of the automatic search engines, of manual searches collectively, and of snowball search, as well as their overlap (for a discussion of these metrics, see page 43). These numbers are absolute; to compute the corresponding relative metrics, divide by the total number of included publications (180); thus, the relative exclusive contribution of snowball search is $(68 \ / \ 180) \times 100\ \% \approx 38\ \%$. The sum of all exclusive contributions is 107 (59 %). I did not consider these metrics during the study; I will assess their implications in Section 6.2.2.

TABLE 4 The overall and exclusive contribution and overlap of the various search modalities

	contrib.			overlap matrix					
	oa.	excl.		S	M	WS	SD	IX	GS
Google Scholar	67	15	GS	38	17	3	7	17	
IEEE Xplore	18	0	IX	12	2	0	0		
ScienceDirect	8	0	SD	5	7	0			
Web of Science	3	0	WS	1	0				
Manual	62	24	M	31					
Snowball	126	68	S						

It is feasible to define a quasi-gold standard (Zhang, Ali Babar, and Tell 2011) by considering all included publications published in one of the publication venues that were targeted by manual search. Unlike Zhang, Ali Babar, and Tell (2011), I include also any such publications found by non-manual searches; this allows me to evaluate the manual searches, as well. In order to compute the QGS, I added to the records of all included publications a tag describing the publication venue; after adding the tags, I checked them by comparing all tags with the corresponding bibliographical data, further, I checked that all journal tags corresponded to journal publications and conference tags to conference publications; I finally checked all tags that were not journal or conference tags individually.

Table 5 shows the quasi-sensitivity of each manual search, computed separately against a QGS consisting only of included publications published in the searched forum itself, and the quasi-sensitivity of manual search overall, computed against the full QGS. Since I do not have reliable numbers on the total number of publications in each manually searched forum, I did not compute specificity for the manual searches. Table 6 shows the quasi-sensitivity and specificity of each automatic search venue, computed against the full QGS.

TABLE 5 The quasi-gold standard and quasi-sensitivity for manual searches

	QGS		
	total	contrib.	q.-s.
ESE	3	1	33 %
CACM	4	4	100 %
TOPLAS	9	9	100 %
LOPLAS	0		
IJMMS	14	12	86 %
IJHCS	2	2	100 %
PPIG	2	1	50 %
ISESE	0		
ESEM	2	2	100 %
OOPSLA	14	9	64 %
SPLASH	0		
ECOOP	13	13	100 %
POPL	3	3	100 %
	66	56	85 %

In computing the QGS and related metrics, I did not limit the Communications of the ACM to years up to 1990 like I did in the manual search. This is a valid simplification, because no included publication was published in the Communications after 1990; this observation also speaks to the validity of restricting the search in the first place.

Finally, an evaluation exercise can be conducted based on the set of secondary studies that have been included in this mapping study, as described in the next section. As described in Section 4.3, the set of relevant primary studies described by the included secondary studies have been extracted. For each such identified primary study, the publications cited by the secondary study in

TABLE 6 The quasi-sensitivity and specificity of automatic searches. The quasi-gold standard consists of all included publications published in the venues for manual search. It consists of 66 individual publications.

		contrib.			
	yield	oa.	QGS	q.-s.	sp.
Google Scholar	8 455	67	16	24 %	1 %
IEEE Xplore	995	18	2	3 %	18 %
ScienceDirect	1 022	8	7	11 %	1 %
Web of Science	20	3	0	0 %	15 %
	10 492	69	18	27 %	1 %

question were recorded. Some of them had not been recorded during searches; disregarding duplicates, altogether 18 publications had been recorded as having been cited by secondary studies without having been recorded during searches. This means that 18 potentially relevant publications had not been found during searches, or if they were found, were thought to be obviously irrelevant. Out of the 2056 publications recorded during searches, this is less than one percent. As a worst case scenario, one might suppose that all of them would have been selected for inclusion had they been found and recorded during search, which means that, hypothetically, 10 % of relevant publications had been missed.

4.2 Selection

Every publication located during the searches was subjected to a three-phase selection decision procedure, summarized in Figure 2. The outcome of each phase was either *exclusion*, in which case the publication did not proceed to the next phase, or *passing*, which allowed the publication to survive that phase and go to the next phase. Passing in Phase III resulted in a provisional decision to include the publication in this mapping study.

4.2.1 Selection criteria

Selection decisions were based on the following seven inclusion and exclusion criteria, written in an interrogatory form. I will generally refer to them as "Question k" or "Qk", where $k = 1, \ldots, 7$:

1. Is this a primary study that attempts to determine the efficacy of a programming language design decision? (If not, skip question 5.)
2. Is this a literature review that attempts to summarize or consolidate research on the efficacy of a programming language design decision? (If not, skip questions 6 and 7.)

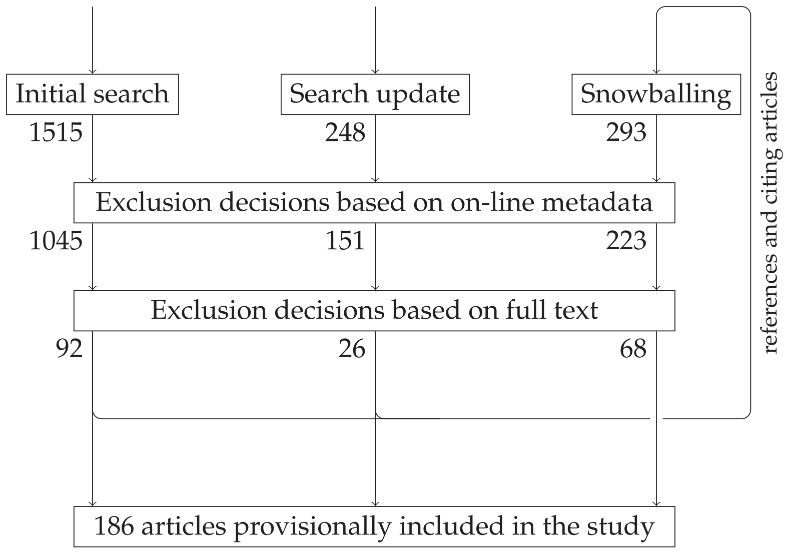

FIGURE 2 Flow diagram of the study selection process. This diagram does not show selection validation and the resulting changes in selection decisions, nor does it show exclusions made *post hoc* during data extraction.

3. Can you find a complete written and published report about this study?[3]
4. Is the study reported in English, Finnish or Swedish?[4]
5. Does this primary study present scientific empirical evidence about their claims?
6. Does this secondary study include any primary studies that present scientific empirical evidence?
7. Does this secondary study discuss scientific empirical evidence in the primary studies under review?

The first two questions are the *inclusion criteria*. The next five questions are the *exclusion criteria*. A publication was excluded if the answer to *both* inclusion criteria or *any one* of the exclusion criteria was negative.

During the search update in early 2013, I added an unnumbered exclusion criterion: any study published after 2012 must be excluded. In some cases, I first passed an article as it had been published online before formal publication but then excluded it once I learned it had later been published formally with a 2013 date.

In interpreting the selection criteria, I used Definitions 1 (on page 14), 3 (on page 19), 2 (on page 14), and 4 (on page 53), as well as the following definition:

3 The "you" in this question addresses the decision-maker, which during Phases I–III was I. Other decision-makers took part in the selection evaluation exercises.

4 I am able to read these languages, and obtaining translations from other languages would not be cost effective in this study. In any case, English is the *lingua franca* of the information technology community, and serious research reports are rarely in other languages.

Definition 6. The *completeness* criterion for study reports requires that the data collection and data analysis (if any) are documented in the report in sufficient detail that there is reason to believe that the reported study could be critically evaluated based on the report alone. Specifically, a mere statement of results is not a "complete" report. This excludes, inter alia, studies that are reported only in lectures, abstracts, extended abstracts and presentation material.

4.2.2 Phases of selection

Phase I of selection took place during searches. I evaluated all publications uncovered by a search based on their title, abstract, keywords and other metadata readily available during the search. I some cases, where it was easily accessible and the available metadata was not very useful, I also briefly looked at the full text. In Phase I, I only applied the inclusion criteria and ignored the exclusion criteria; but I did, on occasion, also exclude in this phase publications that were too short to be able to survive the completeness criterion.

I did not record any exclusion decisions made in Phase I. This was mainly because of the poor specificity of my searches. To counter this, I only excluded in Phase I publications for which this was *obviously* the correct decision; for example, if I felt I needed to explain an exclusion, I passed.

In Phase II, I considered the same online metadata as in Phase I. The main differences between the two phases were that I considered publications in a (literally) random order; that I applied both the inclusion and the exclusion criteria; and that I recorded all exclusions during Phase II, generally with an explanation. The last point allowed me to lower the threshold of exclusion: in Phase II, an exclusion decision required me to be *convinced* that it was the correct decision.

Finally, for Phase III, I attempted to obtain the full text of every publication that had passed Phase II. Failure to obtain it after reasonable effort (which included an interlibrary loan request, unless I judged it obviously futile) was grounds for exclusion under Question 3. I would generally record an explanation for both pass and exclusion decisions. Otherwise, this phase was quite similar to Phase II.

The passing decisions of Phase III amounted to provisional inclusion decisions. Final decisions deviated only in response to problems uncovered during selection validation. A small number of *post hoc* exclusion decisions, modifying the final decisions, occurred during data extraction.

4.2.3 Validation

On December 9, 2011, after Phase II had finished with respect to the first search iteration and once Phase III had resulted in a decision for 150 publications, I selected a sample by the following method:

1. A number n_I between 3 and 7, was randomly chosen. Another number was computed as $n_E = 10 - n_I$.
2. Of the set of publications for which a Phase III inclusion decision had been reached by this time, a subset of n_I publications was randomly chosen.

3. Of the set of publications for which a Phase III exclusion decision had been reached by this time, a subset of n_E publications was randomly chosen.

Thus, the sample consisted of at least three included and at least three excluded publications, the precise ratio of included to excluded publications being randomized, forming a total of 10 publications.

I invited all three of my advisors as well as two of my colleagues to participate in a validation exercise; two (TK and VT) participated. Their task was to make an independent Phase III selection decision for each of the publications in this sample. The procedure for constructing the sample was disclosed to them, but the numbers n_I and n_E were kept confidential.

Pairwise Cohen (1960) kappas and a three-way Fleiss (1971) kappa were computed to assess interrater reliability: between myself and TK, $\kappa = 0.62$ (95 % CI 0.14 to 1.00), between myself and VT, $\kappa = 0.58$ (95 % CI 0.07 to 1.00), between TK and VT, $\kappa = 0.23$ (95 % CI -0.35 to 0.81), and between all three, $\kappa = 0.46$ (95 % CI 0.10 to 0.83). Note that the confidence intervals are of questionable usefulness as the examined publications did not form a simple random sample. On the Landis–Koch verbal scale of strength of agreement (see page 46), all the kappas between myself and the others indicate either a moderate or a substantial strength of agreement. I discussed divergent decisions with all three separately; my original decisions were accepted by all.

After Phase III had finished, on April 30, 2013, I selected a random sequence of 100 publications from among all the 2056 publications recorded during the searches. I asked each of my three advisors to pick the number of publications they would be willing to examine, between 10 and 100. I sent each their chosen number of publications, each an initial subsequence of the sample sequence, and asked them to make independent Phase III selection decisions on each (VL asked for and received some assistance from me, trying to not reveal my own choices; all others were independent). Simultaneously, I re-examined the full sample of 100 publications, making new Phase III selection decisions without reference to my original ones.

Table 7 shows the pairwise Cohen kappas between all the ratings; on the Landis–Koch verbal scale, the strength of agreement was, judging from the point estimates, almost perfect (between AJK-1 and AJK-2, and AJK-2 and VT), substantial (between all others except TK), and fair (between TK and all others). The multi-way Fleiss kappa for all ratings was $\kappa = 0.42$ (95 % CI -0.19 to 1.00, $n = 10$, slight). As the pairwise kappas demonstrate, TK was an outlier in this round; the multi-way Fleiss kappa for all others was $\kappa = 0.77$ (95 % CI 0.02 to 1.00, $n = 10$, substantial).

This exercise concluded with a meeting on August 7, 2013, with I and all my advisors present, in which the divergent decisions were discussed. Altogether eight publications had divergence, and a consensus decision was recorded for all.

Finally, as a *post hoc* validation exercise not considered in the protocol, it is again possible to consider publications cited by included secondary studies. The relevant data is reproduced in Appendix 4. The included secondary studies

TABLE 7 Pairwise Cohen kappas and their 95 % confidence intervals in the second selection validation exercise. AJK-1 is my original set of decisions ($n = 2056$), AJK-2 is my set of re-examinations ($n = 100$), and VT ($n = 28$), TK ($n = 20$), and VL ($n = 10$) are my three supervisors; the pairwise comparisons use the smaller n of the pair, except between TK and VT ($n = 19$).

AJK-2	0.82 (+0.65 to 0.99)			
VT	0.78 (+0.36 to 1.00)	1.00 (+1.00 to 1.00)		
VL	0.62 (−0.10 to 1.00)	0.62 (−0.10 to 1.00)	0.62 (−0.10 to 1.00)	
TK	0.29 (−0.26 to 0.83)	0.38 (−0.15 to 0.92)	0.22 (−0.45 to 0.90)	0.38 (−0.40 to 1.00)
κ	AJK-1	AJK-2	VT	VL

describe altogether 46 primary studies, some of which are duplicates due to the same study being described by multiple secondary studies. They cite 33 publications that have been recorded in the searches; 30 of these have been finally selected for inclusion in this study. Thus, three publications recorded as having been cited by the secondary study (which implies a judgment of mine that they are potentially relevant to the mapping study) were explicitly excluded.

4.3 Data extraction and synthesis

In data extraction and synthesis, I followed the thematic synthesis method as outlined by Cruzes and Dybå (2011a). It is designed for synthesizing evidence from qualitative studies in a systematic review, and thus not all of its features are directly applicable to mapping studies.

4.3.1 A rejected approach

In my initial design of this study, I followed the recommendations of Kitchenham and Charters (2007): I created a data extraction form (reproduced in Appendix 6) and intended to synthesize results from the extracted data.

As described in Subsection 4.2.3, I had performed a selection validation exercise after Phase III had resulted in a decision for 150 publications. After that validation exercise, I conducted a belated pilot extraction on the subset of those 150 publications that had received a Phase III pass, altogether nine publications (with one accompanied by a technical report). As control, my supervisor TK performed an independent extraction.

This approach turned out to be too problematic (as discussed in Section 6.1), and was rejected. I eventually redesigned data extraction and synthesis following the thematic synthesis method as outlined by Cruzes and Dybå (2011a). The rest of this section covers this redesigned method.

4.3.2 Immersion and quote extraction

After the inclusion and exclusion decisions had been finalized, I systematically read every publication selected for inclusion in August and September 2013. This process, referred to as “get[ting] immersed with the data” by Cruzes and Dybå (2011a, p. 276), was time-consuming and mind-numbing given the number of included publications, but taking to heart Cruzes and Dybå’s admonition not to skip this step, it was performed anyway. Afterward, the mapping study protocol was updated to reflect insights up to this point.

Then, in October and early November 2013, I processed each included publication, gathering direct quotes relevant to four topics (design decision, efficacy measurement, research method, and prior studies being followed up on or replicated). At the same time, I grouped publications into studies, combining publications that reported the same study, and splitting a publication if it clearly reported multiple unrelated studies. I assigned each study an identifier of the form Sn, where n is a sequentially assigned number starting from 1; the last identifier assigned was S159.

If a publication reported more than one related study, I split it into sub-studies, coding each separately under the same study identifier. Where necessary to identify a particular sub-study, I use a letter in the sequence $a, b, c \ldots$ to indicate its ordinal within the list of sub-studies under the study identifier.

Some of the studies were secondary studies. Of them, I gathered direct quotes relevant to the secondary study’s overall research method, and identified each primary study it described as a separate sub-study. For each such primary sub-study, I gathered quotes on the four topics listed above as usual. I also identified the publications that the secondary study cites as describing the sub-study.

4.3.3 Coding and post-hoc exclusions

Next, in November 2013, I processed all the studies, identifying relevant ideas related to the three of the four topics mentioned above by assigning labels (also known as codes) describing those ideas to each of the 159 studies and their sub-studies created in the previous step. I developed the code book along the way, by creating a code when needed to code a particular study, splitting a code into two or more codes when its content seemed too broad and so forth. Some of the codes were derived from my *a priori* conceptualizations of the issues, as they applied to the studies at hand; most arose from the studies themselves. I assigned each code to one of three categories, *design decision*, *efficacy*, and *method*, and I required myself to assign at least one code of each category to each sub-study. I also coded each secondary study for its overall method.

The resulting code book is reproduced in Appendix 2, and the code assignments themselves are reproduced in Appendix 3. There are, in total, 245 codes: 178 for coding design decisions (90 coding specific languages, leaving 88 for other uses), 24 for coding facets of efficacy, 40 for coding primary-study methods, and 3 for coding secondary-study methods. While the total is large, the count of codes for efficacy and methods is, in each, within the recommended range of 30–40 codes (Cruzes and Dybå 2011a, Figure 1 and p. 278). The number of design decision codes is large mostly because the codes emerged mostly from the primary studies themselves, and I did not want to prematurely commit to any particular clustering of them.

Some of the studies proved not to have relevant content, revealing a mistake in the decision to include those publications in this study. Those publications and the studies they embody were excluded from this study *post hoc*.

One article (Cartwright 1998) I had initially split into two studies. Study S19 comprised its related works section and was initially intended to be treated as a secondary study. The primary study reported in the same publication was split into S20. Subsequently, during the course of processing all the publications, I made it a rule not to consider related works sections independent secondary studies (allowing for the hypothetical exception of a systematic review reported as a related works section, which never materialized). Accordingly, S19 was excluded *post hoc*. The publication it embodied remains included, as it also reports study S20.

A *post-hoc* validation exercise was attempted in December 2013. I first randomly shuffled all study identifiers and gave the resulting list to my supervisor VT. I asked him to familiarize himself with my code book, and we then discussed any questions and concerns he had developed regarding it. Then, I asked him to independently code as many of the studies he had time for, in the order given by the randomly shuffled list, and using the code book I had developed, without reference to how I had coded them. He coded four:

– For S79 (Iselin 1988), we both assigned the codes Conditionals, COBOL, ProgramComprehension, ControlledExperiment, ProgrammingStudents, ProfessionalProgrammers, and BetweenSubjects. I had, in addition, assigned the codes FeatureDesign, Loops, RandomizedControlledExperiment, and AdvancedProgrammingStudents. VT had also assigned (in parentheses, indicating hesitation) BooleanQueries.
– For S153 (Volos et al. 2009), we both assigned the codes FeatureDesign, STM, NestedParallelism, RuntimePerformance, and BenchmarkPrograms. I had, in addition, assigned the code MemoryLocking. VT had also assigned the code DeterministicParallelism.
– For S115 (Pankratius, Schmidt, et al. 2012), we both assigned the codes LanguageComparison, Scala, Java, ProgrammingEffort, LinesOfCodeComparison, ControlledExperiment, AdvancedProgrammingStudents, and ProfessionalProgrammers. I had, in addition, assigned the codes RandomizedControlledExperiment and WithinSubjects. VT had also assigned the codes ParadigmComparison, Parallelism/Concurrency/Multithreading, BetweenSubjects, LanguageShootout, FP, OOP, RuntimePerformance, PerceivedComplexity, PerceivedIntuitivity, ErrorProneness, and SideEffectingExpressions.
– For S132 (Seixas et al. 2009), we both assigned the codes StaticTyping, DynamicTyping, SecurityVulnerabilityProneness, and CorpusAnalysis. I had

assigned, in addition, the codes FeatureDesign and HistoricalControl. VT had also assigned the codes PHP, Java, C#, VB.net, (OpenCoding?), ParadigmComparison(Type/DynamicType), and SecurityIssuePrevention. The parentheses, solidus, and question mark were VT's own markup.

From these, it is clear that in the detail level the code book was not completely clear. For two out of these four, the differences in the codings would have caused significant differences in their placement in the thematic analysis. I am not aware of a suitable quantitative metric to assess the level of agreement or disagreement in this exercise.

Another *post hoc* validation exercise is simple. Looking at the commit log of my database since commit `3b4f880` dated 26 November 2013, which contained the last regular batch of coding, reveals the following later changes (made during theme development and results drafting) to the assigned codes:

- I added a number of FeatureDesign StaticTyping codes to studies for which I had already assigned particular FeaturePresence codes involving particular static-typing features (at the time thinking that the distinction between feature design and feature presence would be prominent in my thematic model).
- I rearranged the Experiment codes to make explicit that a particular study was nonrandomized or noncontrolled.
- I added the Conditionals code to S143 (Stefik and Gellenbeck 2011).
- I added the FeaturePresence and ProgramIndentation codes to S151 (Vessey and Weber 1984a), due to noticing during writing a since-discarded draft of the results that this coding had been mistakenly omitted.
- I rearranged the codes used to indicate experiment participant background to make it more explicit.

4.3.4 Theme development

In December 2013, I programmed an automatically (re)generable HTML representation of the database collected during this study. It includes most of the raw data in the database, but it also provides generated reports on, for example, codes that occur together. Further, I programmed a query language (see Appendix 1.2) and a method for defining (raw) themes by querying the database. Using this query apparatus, I then proceeded to define a number of raw themes by query, looking for interesting conceptual abstractions within the existing codes.

At the same time, I wrote a draft of the result chapter, looking for a suitable thematic model to present. I drafted a number of bubble plots of various combinations of the data to see if any interesting patterns emerged. A big problem I had with these early drafts was the sheer number of studies to present, and progress was halting as I attempted a narrative linking them all. Eventually, it occurred to me to consider whether all the included studies were of equal worth. I did not have a formal quality appraisal to use, as this was a mapping study, and I had deliberately avoided pre-specifying the research methods that would be allowed in the study (most quality appraisal instruments are rather specific to research approach in the primary studies). Instead, I decided to see if importing a simple evidence hierarchy, which depends only on research method data which I already had, would make the data manageable and reveal interesting patterns.

At a fairly late date it occurred to me to see if there was a pattern in the publication forums; I then proceeded to add a coding for the forums partly for this use and partly to develop the data necessary for Figure 5. Bubble plots cross-tabulating forums and publication years did show a clear pattern, which then suggested a possible interpretation of the data.

The following result chapter is the outcome of these considerations.

5 RESULTS

Overall, 180 publications were finally included in this study; Table 8 lists the publication forums in which at least two included publications have appeared by number of publications; it also gives the tags used to identify forums in a number of subsequent figures. One of them, Figure 3[1] plots publications by forum and publication year, restricted to forums containing at least two included publications. The figure also shows the years each forum was available for publication, which data I gathered mostly from their web sites. I have arranged the publication forums on the y axis to emphasize the rough linear progression that is apparent in the plot that results from some publications no longer publishing these studies and other forums starting such publication; mostly, these starts and stops do not coincide with a forum's birth and death. The one notable exception, the International Journal of Man–Machine Studies, is an artifact of it changing it name at the beginning of 1994 to the International Journal of Human Computer Studies, and not a real coincidence.

Combining publications that report the same study and (in some cases) splitting publications that report more than one study results in the 156 publications listed in Table 9. Each study has been assigned an identifier between S1 and S159; there are three gaps in the identifier list, because of *post hoc* exclusions after identifier assignment. In the list there are 137 primary studies and 19 secondary studies.

Some studies have sub-studies. This usually occurs when a single publication reports several related studies; each of them is allocated a sub-study. For secondary studies, sub-studies encode the primary studies described in the secondary study. All the sub-studies are listed in Appendices 3 and 4, with a letter code appended to a study identifier to distinguish between sub-studies when a study has more than one, and to distinguish between a secondary study and its (primary) sub-studies. There are, in total, 141 sub-studies of primary studies, when considering each study to have at least one sub-study. There are 46 sub-

[1] The figure is a bubble plot, a form of scatterplot in which each data point is shown as a circle whose area is proportional to the data point's magnitude (in this case, the number of studies published in this particular forum in this particular year).

TABLE 8 Publication forums containing at least two included publications, sorted by the number of included publications published in them. The tags are used to identify forums in the bubble plots involving forums.

Forum	#	Tag
International Journal of Man–Machine Studies	14	j:IJMMS
ACM SIGPLAN International Conference on Object-Oriented Systems, Languages, and Applications	14	proc:OOPSLA
European Conference on Object-Oriented Programming	13	proc:ECOOP
ACM Transactions on Programming Languages and Systems	9	j:TOPLAS
Technical reports published by various institutions	7	tr
IEEE Transactions on Software Engineering	6	j:TSE
International Conference on Software Engineering	5	proc:ICSE
Communications of the ACM	4	j:CACM
Journal of Systems and Software	4	j:JSS
Workshop on Evaluation and Usability of Programming Languages and Tools	4	proc:PLATEAU
Empirical Software Engineering	3	j:ESE
Software: Practice and Experience	3	j:SPE
IEEE International Conference on Software Maintenance	3	proc:ICSM
ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages	3	proc:POPL
The Computer Journal	2	j:CJ
International Journal of Human-Computer Studies	2	j:IJHCS
Journal of Occupational Psychology	2	j:JOP
ACM SIGCSE Bulletin	2	j:SIGCSEB
Software Quality Journal	2	j:SQJ
AFIPS National Computer Conference	2	proc:AFIPS
ACM CHI Conference on Human Factors in Computing	2	proc:CHI
ACM/IEEE International Symposium on Empirical Software Engineering and Measurement	2	proc:ESEM
IEEE International Conference on Program Comprehension	2	proc:ICPC
Psychology of Programming Annual Conference	2	proc:PPIG
Simpósio Brasileiro de Engenharia de Software	2	proc:SBES
ACM Technical Symposium on Computer Science Education	2	proc:SIGCSE
ACM Symposium on Parallelism in Algorithms and Architectures	2	proc:SPAA
arXiv	2	arXiv
Bachelor’s theses in various universities	2	thesis:BSc

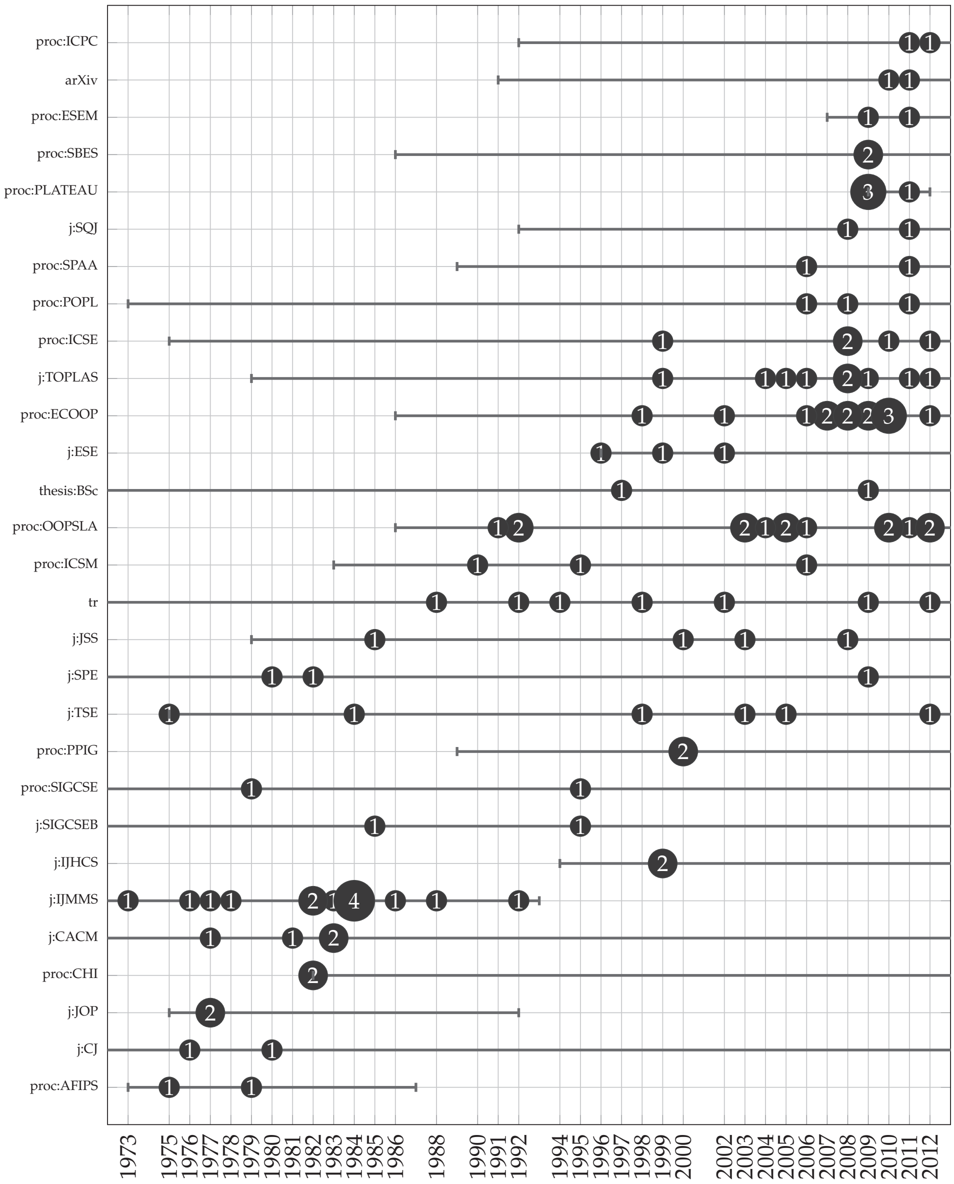

FIGURE 3 Bubble plot of included publications by publication forum and publication year, restricted to forums containing at least two included publications. The years when the forum has been available are indicated.

TABLE 9 Included studies

Study	P/S	Consists of	Study	P/S	Consists of	Study	P/S	Consists of
S1	P	Ahmad and Talha 2002	S54	P	Fähndrich and Leino 2003	S107	P	Necula et al. 2005
S2	P	Ahsan et al. 2009	S55	P	Gannon and Horning 1975a,b; Gannon 1976	S108	P	Norcio 1982
S3	P	Aldrich et al. 2002	S56	P	Gannon 1977			(S109 EXCLUDED)
S4	P	Andreae et al. 2006	S57	P	Gil and Shragai 2009	S110	P	Nystrom et al. 2006
S5	S	Arblaster 1982	S58	P	Gil and Lenz 2010	S111	P	Nyström et al. 2007
S6	P	Badreddin, Forward, et al. 2012; Badreddin and Lethbridge 2012	S59	P	Gilmore and Green 1984	S112	S	Pane and Myers 2000
S7	P	Badreddin and Lethbridge 2012	S60	P	Green 1977	S113	S	Pane and Myers 2006
S8	P	Badri et al. 2012	S61	S	Green 1980	S114	P	Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011
S9	P	Barnes and Welch 2001	S62	P	Greenwood et al. 2007	S115	P	Pankratius, Schmidt, et al. 2012
S10	P	Bartsch and Harrison 2008	S63	P	Halverson 1993	S116	P	Patel and Gilbert 2008
S11	P	Benander and Benander 1997	S64	P	Hanenberg, Kleinschmager, and Josupeit-Walter 2009; Kleinschmager 2009	S117	P	Patterson 1981
S12	P	Benton et al. 2004	S65	P	Hanenberg 2009, 2010a,b	S118	P	Perrott et al. 1980
S13	P	Biermann et al. 1983	S66	P	Harel and McLean 1985	S119	P	Poletto et al. 1999
S14	P	Bocchino et al. 2011	S67	P	Harrison, Smaraweera, et al. 1996	S120	P	Prechelt and Tichy 1996, 1998
S15	S	Boehm-Davis 2002	S68	P	Harrison, Counsell, et al. 2000	S121	P	Prechelt 2000; Prechelt 2003
S16	S	Briand et al. 1999	S69	P	Henry and Humphrey 1988; Henry and Humphrey 1990; Henry and Humphrey 1993	S122	P	Prechelt, Unger, et al. 2003; Unger and Prechelt 1998
S17	P	Burckhardt et al. 2011	S70	P	Hertz and Berger 2005	S123	P	Przybyłek 2011
S18	P	Cacho et al. 2009	S71	P	Hicks et al. 2004	S124	P	Qi and Myers 2010
		(S19 EXCLUDED)	S72	P	Hitz and Hudec 1995	S125	P	Ramalingam and Wiedenbeck 1997
S20	P	Cartwright 1998	S73	S	Hoc 1983	S126	S	Roberts 1995
S21	P	Castor, Cacho, et al. 2009	S74	P	Hochstein and Basili 2006; Hochstein, Basili, et al. 2008	S127	P	Rossbach et al. 2009, 2010
S22	P	Castor, Oliveira, et al. 2011	S75	P	Hoffman and Eugster 2008	S128	P	Saal and Weiss 1977
S23	P	Cesarini et al. 2008	S76	P	Hu et al. 2010	S129	S	Sadowski and Shewmaker 2010
S24	P	Chalin and James 2007	S77	P	Huang and Smaragdakis 2011	S130	P	Sawadpong et al. 2012
S25	P	Champeaux et al. 1992	S78	P	Hudak and Jones 1994	S131	P	Scholte et al. 2012
S26	P	Charles et al. 2005	S79	P	Iselin 1988	S132	P	Seixas et al. 2009
S27	P	Chen and Vecchio 1992	S80	P	Jim et al. 2002	S133	S	Sheil 1981
S28	P	Cherry 1986	S81	S	Johnson 2002	S134	P	Sheppard et al. 1979
S29	P	Coelho et al. 2008	S82	P	Kesler et al. 1984	S135	S	Shneiderman 1975
S30	P	Cohen et al. 2012	S83	P	Kleinschmager et al. 2012; Kleinschmager 2012	S136	P	Shneiderman 1976; Shneiderman and Mayer 1979
S31	P	Condit et al. 2003	S84	P	Klerer 1984	S137	P	Sime et al. 1973, 1999
S32	S	Curtis 1982	S85	P	Kosar et al. 2010	S138	P	Sime et al. 1977
S33	P	Daly et al. 1995; Daly et al. 1996	S86	P	Kulesza et al. 2006	S139	S	Sime, Arblaster, et al. 1977
S34	P	Daly, Sazawal, et al. 2009	S87	S	Laughery and Laughery 1985	S140	P	Smith and Dunsmore 1982
S35	S	Deligiannis et al. 2002	S88	P	Leblanc and Fischer 1982	S141	P	Soloway et al. 1983
S36	P	Demsky and Dash 2008	S89	P	Lee et al. 2003	S142	P	Stefik, Siebert, et al. 2011
S37	P	Dolado et al. 2003	S90	P	Lewis et al. 1991, 1992	S143	P	Stefik and Gellenbeck 2011
S38	P	Dolby et al. 2012	S91	P	Lima et al. 2011	S144	P	Stuchlik and Hanenberg 2011
S39	P	Doscher 1990	S92	P	Liu et al. 2006	S145	P	Taveira et al. 2009
		(S40 EXCLUDED)	S93	P	Lucas and Kaplan 1976	S146	P	Tenny 1985
S41	P	Dyer et al. 2012	S94	P	Luff 2009	S147	P	Thies and Amarasinghe 2010
S42	P	Ebcioğlu et al. 2006	S95	P	Madeyski and Szala 2007	S148	P	Tobin-Hochstadt and Felleisen 2008
S43	P	Embley 1978	S96	P	Malayeri and Aldrich 2009	S149	P	Tonella and Ceccato 2005
S44	P	Endrikat and Hanenberg 2011	S97	P	Mayer et al. 2012b; Mayer et al. 2012a	S150	P	Valente et al. 2010
S45	P	Engebretson and Wiedenbeck 2002	S98	P	McCaffrey and Bonar 2010	S151	P	Vessey and Weber 1984a
S46	P	Ertl 1999	S99	P	McEwan et al. 2010	S152	S	Vessey and Weber 1984b
S47	P	Ferrari et al. 2010	S100	P	McIver 2000	S153	P	Volos et al. 2009
S48	P	Ferrett and Offutt 2002	S101	P	Miara et al. 1983	S154	P	Walker, Bamassad, et al. 1998; Walker, Baniassad, et al. 1999
S49	P	Figueiredo et al. 2008	S102	P	Millstein 2004; Millstein et al. 2009	S155	P	Walker, Lamere, et al. 2002
S50	P	Flanagan et al. 2008	S103	P	Mortensen et al. 2012	S156	P	Weimer and Necula 2008
S51	P	Foster et al. 2006	S104	P	Myers, Giuse, et al. 1992	S157	P	Westbrook et al. 2012
S52	S	Furuta and Kemp 1979	S105	P	Myrtveit and Stensrud 2008	S158	P	Wiedenbeck and Ramalingam 1999
S53	S	Fyfe 1997b,a	S106	P	Nanz et al. 2010; Nanz et al. 2011b,a	S159	P	Wiedenbeck, Ramalingam, et al. 1999

P = primary study
S = secondary study

studies of secondary studies.

I have used the secondary studies in the validation of the search and selection processes (see Subsections 4.1.4 and 4.2.3). I will only consider primary studies from now on.

5.1 Thematic model

The thematic model is the kernel of the results of the study; from it flow all the answers to the research questions, and any *post hoc* observations that can be made. The set of primary studies in this mapping study has three *a priori* thematic dimensions that follow from the research questions. Each study has been coded on the *design decisions* and on the *facets of efficacy* it investigates, as well as the *research method* it uses. Each code has also been assigned a subcategory within these three dimensions; some of the thematic model is specified using the subcategories. The codes used, including their subcategories, are listed in Appendix 2, and the code assignments are given in Appendix 3.

5.1.1 Periphery

The process used to select studies for inclusion in this mapping study was deliberately designed to include a study if there was doubt. This implies that at least some of the included studies are questionable from the point of view of this mapping study. The first task of the thematic model is to identify categories of these questionable studies; I will call them the *periphery*.

There are a number of included primary studies that merely compare languages or, through such a comparison, attempt to evaluate paradigms or language generations, without any attempt to isolate particular features for study. They are identifiable by having been coded as LanguageComparison, GenerationComparison, or ParadigmComparison without a FeatureDesign or FeaturePresence code; there are also one or more codes with subcategory SpecificLanguage, LanguageGeneration, or Paradigm that identify the languages, generations, and paradigms under comparison. Such studies are potentially of some interest to language designers, but they are likely to be fairly uninformative.

The most common language comparison pair is AspectJ and Java (12 sub-studies); the following language pairs have two comparison sub-studies each: C and C++, C and CCured, C and Pascal, C++ and Pascal, and Java and Umple. Fifteen sub-studies have claimed to compare the object-oriented and aspect-oriented paradigms, seven sub-studies have claimed to compare the object-oriented and procedural paradigms; one sub-study each has claimed to compare object-oriented programming to functional programming, system programming languages to scripting languages, and declarative paradigm to the procedural paradigm. One sub-study has claimed to compare the third generation to the fourth generation.

There are also 15 studies (and as many sub-studies), coded BenchmarkPrograms, whose research method is to select programs or programming problems from the literature or folklore and to demonstrate that the design decision under study is capable of dealing with them. It is arguable that this is not empirical at all under the definition I have adopted. In any case, I will not consider them further.

5.1.2 Core

The remaining studies, that is, those primary studies that are coded FeatureDesign or FeaturePresence and are not coded BenchmarkPrograms, form the *core*. It consists of 63 studies and 65 sub-studies.

Figure 4 shows a version of Figure 3 restricted to core publications only; I have again chosen the order of the forums to emphasize the pattern that is apparent in the plot, similar to the one for all publications.

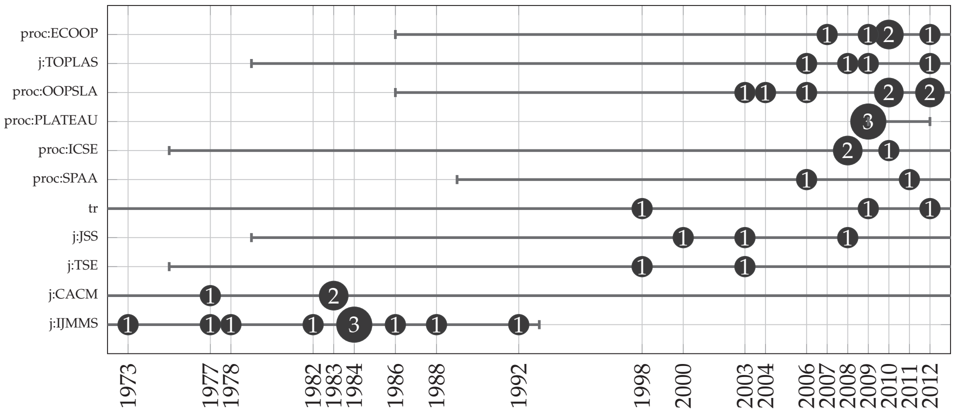

FIGURE 4 Bubble plot of included core publications by publication forum and publication year, restricted to forums containing at least two included core publications. Note that absent years in this plot do not necessarily signify publication-less years. The years when each forum has been available are indicated.

In an abuse of metaphors, the core may be further analyzed as an onion, based on a hierarchy of evidence: the *inner core* consists of randomized controlled experiments; there are middle layers for non-randomized controlled experiments and other experiments; and the outer core consists of non-experimental studies.

There are 22 studies (and as many sub-studies) in the inner core. As discussed above, they are randomized controlled experiments that do not merely compare languages, paradigms, or language generations. Four of them were published in the International Journal of Man–Machine Studies, three as technical reports, and two in the Journal of Systems and Software. A number of forums published only one study; of course, some studies were published in more than one forum. Tables 10 and 11 summarize the design decisions and facets of efficacy, respectively, that these studies investigate.

TABLE 10 Design decisions investigated by randomized controlled experiments in the core.

Design decisions	Studies
Static versus dynamic typing	S34 (Daly, Sazawal, et al. 2009)
	S83 (Kleinschmager 2012; Kleinschmager et al. 2012)
	S97 (Mayer et al. 2012a; Mayer et al. 2012b)
	S144 (Stuchlik and Hanenberg 2011)
Class inheritance	S20 (Cartwright 1998)
	S33 (Daly et al. 1995; Daly et al. 1996)
	S68 (Harrison, Counsell, et al. 2000)
	S122 (Unger and Prechelt 1998; Prechelt, Unger, et al. 2003)
Software transactional memory	S22 (Castor, Oliveira, et al. 2011)
	S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011)
	S127 (Rossbach et al. 2009, 2010)
Conditionals	S59 (Gilmore and Green 1984)
	S63 (Halverson 1993)
	S79 (Iselin 1988)
Program indentation	S82 (Kesler et al. 1984)
	S108 (Norcio 1982)
Fixity	S28 (Cherry 1986)
Task-specific constructs	S45 (Engebretson and Wiedenbeck 2002)
Loops	S79 (Iselin 1988)
GOTO	S93 (Lucas and Kaplan 1976)
Java- vs Eiffel-style concurrency	S106 (Nanz et al. 2010; Nanz et al. 2011a,b)
Static versus no typing	S120 (Prechelt and Tichy 1996, 1998)
Comments	S134 (Sheppard et al. 1979)
Structured programming	S134 (Sheppard et al. 1979)

TABLE 11 Facets of efficacy studied by randomized controlled experiments in the core.

Facet of efficacy	Studies
Programming effort	S22 (Castor, Oliveira, et al. 2011) S28 (Cherry 1986) S83 (Kleinschmager 2012; Kleinschmager et al. 2012) S93 (Lucas and Kaplan 1976) S97 (Mayer et al. 2012a; Mayer et al. 2012b) S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) S127 (Rossbach et al. 2009, 2010) S144 (Stuchlik and Hanenberg 2011)
Program comprehension	S20 (Cartwright 1998) S59 (Gilmore and Green 1984) S68 (Harrison, Counsell, et al. 2000) S79 (Iselin 1988) S82 (Kesler et al. 1984) S106 (Nanz et al. 2010; Nanz et al. 2011a,b) S134 (Sheppard et al. 1979)
Error proneness	S22 (Castor, Oliveira, et al. 2011) S28 (Cherry 1986) S63 (Halverson 1993) S106 (Nanz et al. 2010; Nanz et al. 2011b,a) S120 (Prechelt and Tichy 1996, 1998) S122 (Unger and Prechelt 1998; Prechelt, Unger, et al. 2003) S127 (Rossbach et al. 2009, 2010)
Maintenance effort	S20 (Cartwright 1998) S33 (Daly et al. 1995; Daly et al. 1996) S45 (Engebretson and Wiedenbeck 2002) S93 (Lucas and Kaplan 1976) S122 (Unger and Prechelt 1998; Prechelt, Unger, et al. 2003) S134 (Sheppard et al. 1979)
Debugging effort	S34 (Daly, Sazawal, et al. 2009) S106 (Nanz et al. 2010; Nanz et al. 2011a,b)
Lines-of-code comparison	S22 (Castor, Oliveira, et al. 2011) S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011)
Performance in a Cloze test	S108 (Norcio 1982)
Modifiability	S68 (Harrison, Counsell, et al. 2000)

The first layer on top of the inner core consists of core studies that are controlled experiments but have not randomized their allocation of participants to groups (or if they have, they did not report it). This layer consists of 13 studies (and as many sub-studies), four of which were published in the International Journal of Man–Machine Studies, three in the Communications of the ACM, two in the PLATEAU conference; again, a number of forums published only one study. Table 12 and 13 list the design decisions and facets of efficacy, respectively, investigated by the inner core and the first layer together (that is, by all controlled experiments whether or not they are randomized).

TABLE 12 Design decisions investigated by controlled experiments in the core, adding nonrandomized experiments to the categories of Table 10 and new categories.

Design decisions	Studies
Conditionals	3 randomized controlled experiments, and S43 (Embley 1978) S136 (Shneiderman 1976; Shneiderman and Mayer 1979) S137 (Sime et al. 1973, 1999) S138 (Sime et al. 1977) S151 (Vessey and Weber 1984a)
Static versus dynamic typing	4 randomized controlled experiments, and S65 (Hanenberg 2009, 2010a,b)
Class inheritance	4 randomized controlled experiments
Software transactional memory	3 randomized controlled experiments, and S94 (Luff 2009)
Program indentation	2 randomized controlled experiments, and S101 (Miara et al. 1983) S151 (Vessey and Weber 1984a)
Loops	1 randomized controlled experiment, and S43 (Embley 1978) S141 (Soloway et al. 1983)
Interprocess message passing	no randomized controlled experiments, and S74 (Hochstein and Basili 2006; Hochstein, Basili, et al. 2008) S94 (Luff 2009)
Static versus no typing	1 randomized controlled experiment, and S56 (Gannon 1977)
Comments	1 randomized controlled experiment, and S146 (Tenny 1985)
Fixity	1 randomized controlled experiment
Task-specific constructs	1 randomized controlled experiment
GOTO	1 randomized controlled experiment
Structured programming	1 randomized controlled experiment
Java- vs Eiffel-style concurrency	1 randomized controlled experiment
Side-effects in expressions	no randomized controlled experiments, and S37 (Dolado et al. 2003)
Nested subroutines	no randomized controlled experiments, and S146 (Tenny 1985)

The second layer consists of non-controlled experiments. Such studies do

TABLE 13 Facets of efficacy studied by controlled experiments in the core, building up on Table 11.

Facet of efficacy	Studies
Error proneness	7 randomized controlled experiments, and
	S56 (Gannon 1977)
	S65 (Hanenberg 2009, 2010a,b)
	S74 (Hochstein and Basili 2006; Hochstein, Basili, et al. 2008)
	S137 (Sime et al. 1973, 1999)
	S138 (Sime et al. 1977)
	S141 (Soloway et al. 1983)
	S151 (Vessey and Weber 1984a)
Programming effort	8 randomized controlled experiments, and
	S65 (Hanenberg 2009, 2010a,b)
	S74 (Hochstein and Basili 2006; Hochstein, Basili, et al. 2008)
	S94 (Luff 2009)
	S137 (Sime et al. 1973, 1999)
	S151 (Vessey and Weber 1984a)
Program comprehension	7 randomized controlled experiments, and
	S37 (Dolado et al. 2003)
	S43 (Embley 1978)
	S101 (Miara et al. 1983)
	S136 (Shneiderman 1976; Shneiderman and Mayer 1979)
	S146 (Tenny 1985)
Maintenance effort	6 randomized controlled experiments
Lines-of-code comparison	2 randomized controlled experiments, and
	S94 (Luff 2009)
Debugging effort	2 randomized controlled experiments
Performance in a Cloze test	1 randomized controlled experiment
Modifiability	1 randomized controlled experiment
Perceived complexity	no randomized controlled experiments, and
	S94 (Luff 2009)

attempt to control one or more variables in order to influence one or more other variables (and therefore qualify as experiments), but they do not allocate their participants into groups to cover all relevant values of the independent variables, and, in the case of a repeated-measures design, to cover all relevant ways to sequence the dependent-variable measurements. There are five such studies (and as many sub-studies), two of which were published in the International Journal of Man–Machine studies, one in the Journal of Occupational Psychology, one in the ICSE conference, and one in the International Conference on Aspect-oriented Software Development. Tables 14 and 15 summarize the design decisions and facets of efficacy, respectively, studied in the non-controlled experiments.

TABLE 14 Design decisions investigated by non-controlled experiments in the core

Design decisions	Studies
Conditionals	8 controlled experiments, and S27 (Chen and Vecchio 1992) S60 (Green 1977) S140 (Smith and Dunsmore 1982)
Loops	3 controlled experiments, and S140 (Smith and Dunsmore 1982)
Structured programming	1 controlled experiment S140 (Smith and Dunsmore 1982)
Pointcuts	no controlled experiments, and S41 (Dyer et al. 2012)
Conditional compilation	no controlled experiments, and S49 Figueiredo et al. 2008

TABLE 15 Facets of efficacy studied by non-controlled experiments in the core

Facet of efficacy	Studies
Program comprehension	12 controlled experiments, and S27 (Chen and Vecchio 1992) S60 (Green 1977) S140 (Smith and Dunsmore 1982)
Program quality	no controlled experiments, and S41 (Dyer et al. 2012)
Design stability	no controlled experiments, and S49 Figueiredo et al. 2008

The third and outer layer of the core consists of all other studies, 23 in total (containing 24 sub-studies). Five were published in the OOPSLA conferences, four in the ECOOP conferences, and two in the ICSE conference; a number of other forums published one each. Figures 16 and 17 summarize the design decisions and facets of efficacy, respectively, investigated by at least two core studies; additionally, nine studied particular features of static type systems and two studied particular features of shared-memory communication. Figure 5 cross-tabulates the facets of efficacy and primary research method used in each sub-study in this outer layer.

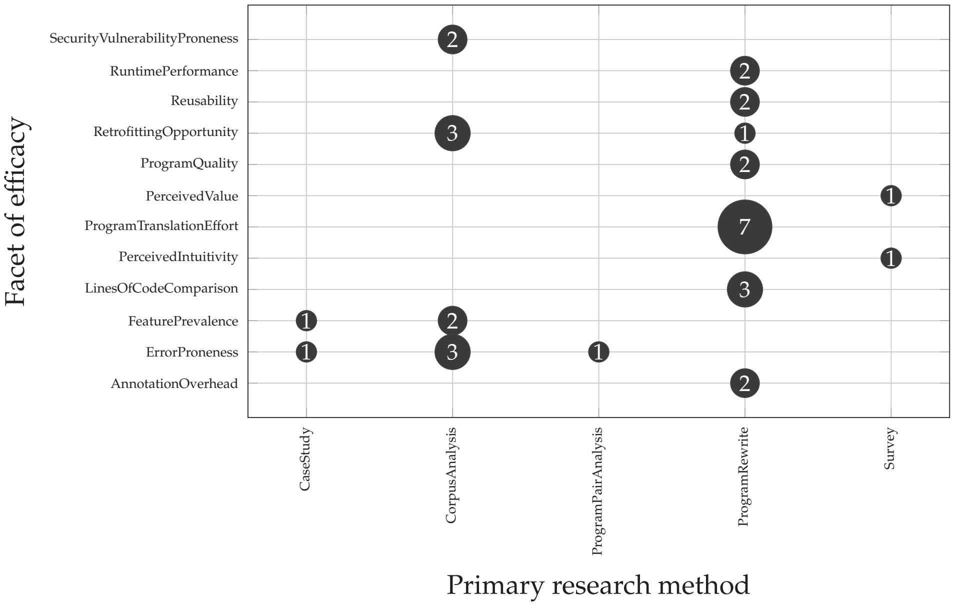

FIGURE 5 Bubble plot of core sub-studies, excluding experiments, categorized by the facets of efficacy used and the primary research methods used.

TABLE 16 Design decisions investigated by core studies, adding non-experiments to the categories of Tables 12 and 14, adding new categories, and removing all categories with only one core study

Design decisions	Studies
Conditionals	11 experiments, and S143 (Stefik and Gellenbeck 2011)
Static versus dynamic typing	5 experiments, and S132 (Seixas et al. 2009)
Loops	4 experiments, and S143 (Stefik and Gellenbeck 2011)
Class inheritance	4 experiments
Software transactional memory	4 experiments
Program indentation	4 experiments
Pointcuts	1 experiment, and S47 (Ferrari et al. 2010)
Static structural subtyping	no experiments, and S89 (Lee et al. 2003) S96 (Malayeri and Aldrich 2009)
Interprocess message passing	2 experiments
Static versus no typing	2 experiments
Comments	2 experiments
Structured programming	2 experiments

TABLE 17 Facets of efficacy studied by at least three core studies, building up on Tables 13 and 15.

Facet of efficacy	Studies
Error proneness	14 experiments, and S47 (Ferrari et al. 2010) S88 (Leblanc and Fischer 1982) S130 (Sawadpong et al. 2012) S131 (Scholte et al. 2012) S132 (Seixas et al. 2009)
Program comprehension	15 experiments
Programming effort	13 experiments
Program translation effort	no experiments, and S30 (Cohen et al. 2012) S38 (Dolby et al. 2012) S92 (Liu et al. 2006) S102 (Millstein 2004; Millstein et al. 2009) S110 (Nystrom et al. 2006) S156 (Weimer and Necula 2008) S157 (Westbrook et al. 2012)
Lines-of-code comparison	3 experiments, and S92 (Liu et al. 2006) S102 (Millstein 2004; Millstein et al. 2009) S124 (Qi and Myers 2010)
Maintenance effort	6 experiments
Retrofitting opportunity	no experiments, and S24 (Chalin and James 2007) S51 (Foster et al. 2006) S57 (Gil and Shragai 2009) S96 (Malayeri and Aldrich 2009)
Feature prevalence	no experiments, and S58 (Gil and Lenz 2010) S89 (Lee et al. 2003) S147 (Thies and Amarasinghe 2010)
Program quality	1 experiment, and S75 (Hoffman and Eugster 2008) S102 (Millstein 2004; Millstein et al. 2009)

5.1.3 Temporal pattern

Figure 6 shows how included publications distribute between years. The earliest publication dates from 1973, and the latest from 2012 (the cutoff year for this mapping study), giving 40 years of publications, an average of 4.5 publications per year. A pattern is quite clear in this figure: there are peaks in publications in 1977, 1982–1984, 1992, 1999, 2002, 2006, and from 2008 onward; also, the number of publications per year has increased dramatically first in 1999, and then from 2008 onward.

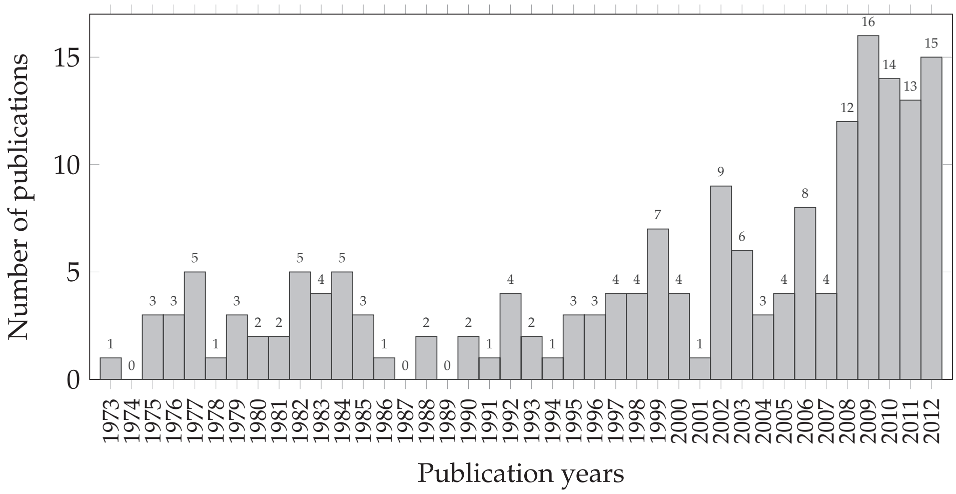

FIGURE 6 The number of included publications per year

Figure 7 summarizes the years of first publication of all primary studies. The same pattern as noted with Figure 6 is visible here too: peaks in 1977, 1982–1984, 1992, 1999, 2002, 2006, and from 2008 onward, and a dramatic rise in the number of studies per year first in 2002 (not 1999) and from 2008 onward.

Figure 8 shows the distribution of first publications of core studies over the years. The average rate is 1.6 new core studies published per year. Again, the pattern seen with Figures 6 and 7 is visible, though with minor mutations: peaks in 1977, 1982–1984, 1998 (not 1999), and from 2008 onward, and a dramatic rise in the number of studies per year from 2008 onward. The peaks of 1992, 2002, and 2006 disappear; instead, the dramatic rise in the recent years is preceded by a moderate rise from 2003 onward.

Figure 9 shows the distribution of the publication years of the inner core. The average rate is 0.6 studies per year. The patterns seen with Figures 8, 7 and 6 is muted but partially still visible: peaks in 1984, 1998, and 2009. The rise after 2009 is not so dramatic, but it is noticeable: an average of 2 inner core studies were first published per year in 2009–2012. A notable change is the appearance of long gaps, 1989–1992 and 2003–2008.

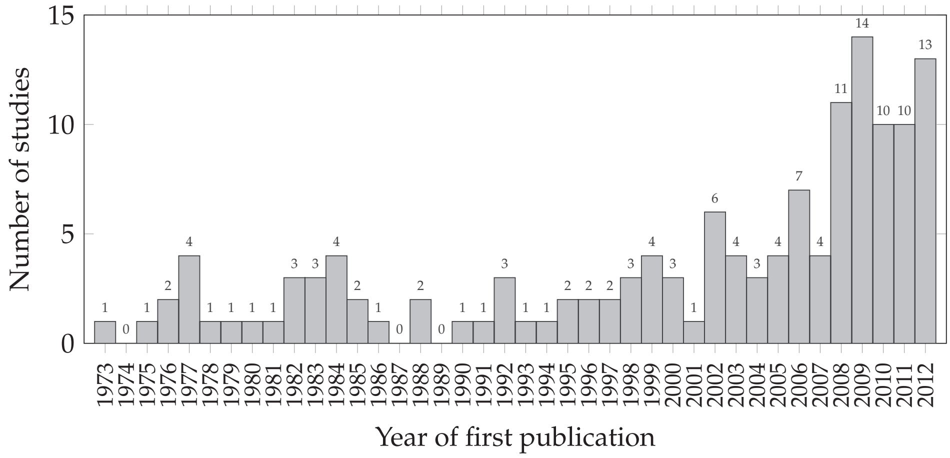

FIGURE 7 The number of included primary studies per publication year

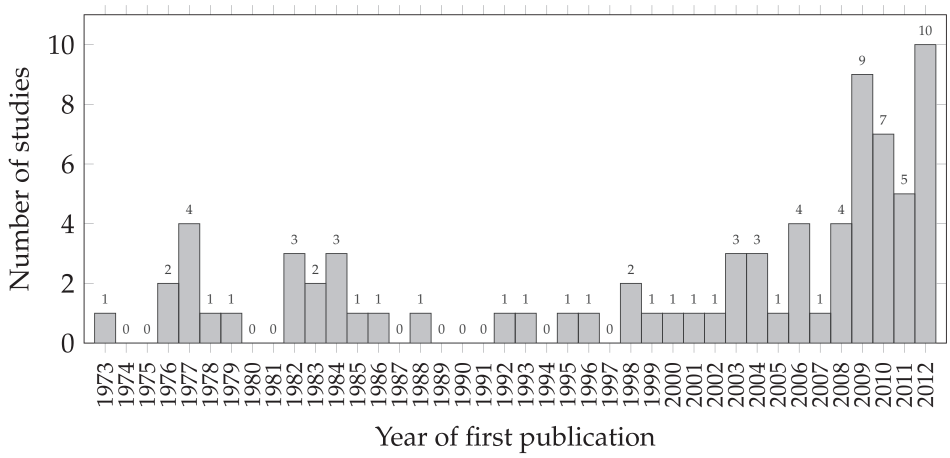

FIGURE 8 The number of included core studies per publication year

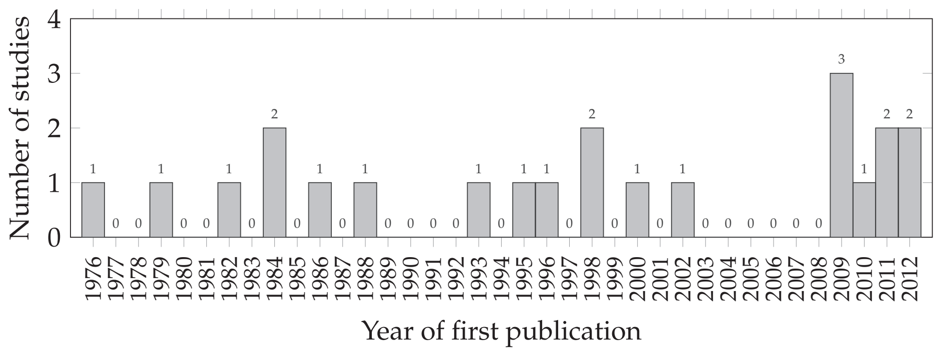

FIGURE 9 The number of randomized controlled experiments in the core per publication year

5.2 Answers to research questions

I will now turn to answering each of the research questions.

RQ1 *How much has the efficacy of particular programming language design decisions been empirically studied?*

In this study, I have identified 65 core sub-studies of primary studies spanning four decades, between 1973–2012, each studying the efficacy of some language design decision empirically. There were 141 sub-studies in all, including the periphery. If one were to consider only the traditional gold standard of efficacy evidence – randomized controlled experiments – there still are 22 core sub-studies, the earliest dating from 1976. For the last category, there is a noticeable gap in publications between 1989–1992 and 2003–2008.

RQ2 *Which programming language design decisions have been studied empirically for efficacy?*

The *form of the conditional statement* is the most studied design decision in the core, with altogether 11 core experiments, including 8 controlled experiments, of which 3 were randomized. As can be seen in Figure 10, this design decision has been studied over a long period of time. It is the one studied by the oldest study (Sime et al. 1973, 1999), and it has been studied nearly up to the present day (Stefik and Gellenbeck 2011), though there was a long gap after the Halverson (1993) randomized controlled experiment. About half of these studies have concentrated on comparing the styles defined by Sime et al. (1973, 1977, 1999), namely, JUMP, NEST, NEST-BE, and NEST-INE, discussed in Subsection 2.4.1.

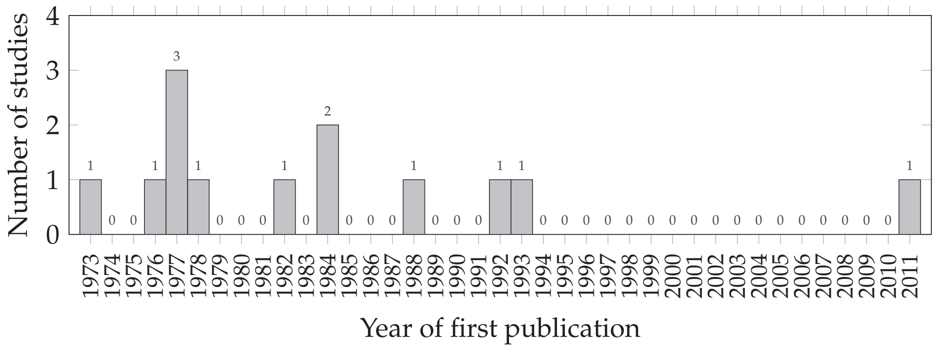

FIGURE 10 The number of core studies of conditionals per year

The choice between static and dynamic typing is the second most studied design decision in the core, with altogether 6 core studies, of which 5 were controlled experiments, of which 4 were randomized. The oldest of these studies are those of Daly, Sazawal, et al. (2009) and Seixas et al. (2009); it is, therefore a very new area of research, even though the design options themselves date from the 1960s.

I do not include Prechelt (2000) and Prechelt (2003), as they are pure language comparisons and thus in the periphery, nor Gannon (1977) and Prechelt and Tichy (1996, 1998), as they compare static typing to the lack of type checking altogether.

Loops are the third studied design decision in the core, with 5 core studies, of which one was a non-experiment, one a non-controlled experiment, one a randomized controlled experiment, and two non-randomized controlled experiments. The oldest is the controlled experiment of Embley (1978), studying the KAIL selector for both loops and conditionals, and the newest is Stefik and Gellenbeck (2011), investigating the syntactic options for many different language constructs. Figure 10 shows the distribution of the studies over the years.

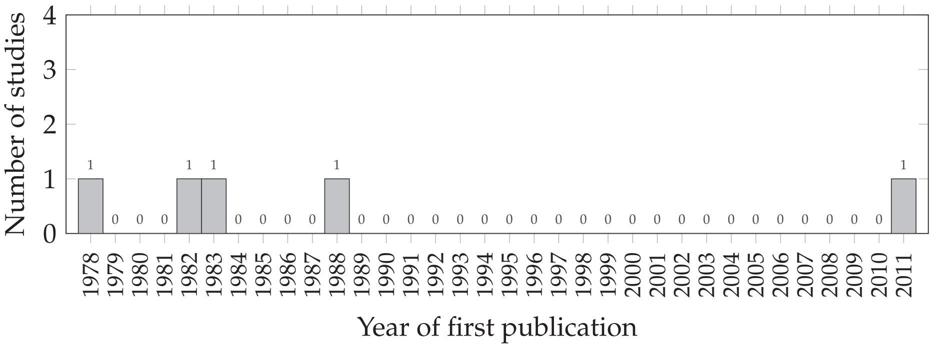

FIGURE 11 The number of core studies of loops per year

The full list of all design decisions with at least two core studies is given in Table 16.

RQ3 *Which facets of efficacy regarding programming language design decisions have been studied empirically?*

As seen in Table 17, the top three facets of efficacy in core studies are *error proneness* (measured typically by seeing how many errors participants make), *program comprehension*, and *programming effort* (measured typically by wall-clock time required to complete an experimental task).

Some of the other facets of efficacy identified in Table 17 may need some explanation. For example, *program translation effort* occurs, as can seen in Figure 5, only with the program-rewrite method. What these two mean is illustrated by the following quote from Westbrook et al. (2012, p. 633–634):

> "More specifically, we have taken a set of HJ programs, written without permissions in mind, and ported them to HJp by adding enough annotations to statically guarantee race-freedom. [. . .] We measured the number of lines of code (LoC) that had to be modified (from the HJ version) to statically ensure race-freedom."

Thus, efficacy is measured by seeing how much effort (proxied here by the number of lines of code modified) it takes to convert existing programs to the new language feature.

Another perhaps-not-clear facet is *retrofitting opportunity*. It is illustrated by the following quote from Malayeri and Aldrich (2009, p. 109):

> "In summary, we found that a number of different aspects of Java programs suggest the potential utility of structural subtyping. While some of the results are not as strong as others, taken together the data suggests that programs could benefit from the addition of structural subtyping, even if they were written in a nominally-typed language."

What is measured here is how well a new feature would fit an existing language, based on the usage patterns actually extant in real-world code in that language; in other words, the degree of opportunity to retrofit the feature to the language. *Feature prevalence* is similar, measuring how much a particular feature is in use in real-world code, which information may be useful to a designer considering its modification in a language, or its introduction to a new language.

RQ4 *Which empirical research methods have been used in studying the efficacy of particular programming language design decisions?*

Among the core sub-studies, there are 41 *experiments*, 11 *program rewrite studies*, 8 *program corpus analyses*, 2 *case studies*, 2 *surveys*, and 1 *program pair analysis*. I have explained what I mean by experiments and program rewrite studies earlier. Program corpus analysis consists of analyzing without modifying a (usually large) set of programs written for other purposes than the study in question. I use the term "case study" consistent with the Yinite[2] definition (Yin 2009; Runeson et al. 2012). Surveys refer to questionnaire-based research. A program pair analysis consists of taking a small number of pairs of related programs not written specifically for this study and comparing them.

Of the experiments in the core, 18 are *between subjects* and 14 are *within subjects*; it is not possible to assign 9 experiments to either category. One (non-controlled) experiment in the core does not have human participants. A total of 35 experiments in the core use *programmers* of various kinds as participants, and 5 use *non-programmers*. Of the 35 programmer experiments, 29 use *students*, 7 used *professional* programmers, and one uses *end-user programmers* (some used participants from more than one of these groups). Of the 29 programming student experiments, 5 use *beginners* (students who are taking or have completed basic programming courses but no more), while 15 use *advanced students*; for 9 student experiments, it is not possible to determine the student type.

RQ5 *How common are follow-up or replication studies, either by the original researchers or by others?*

Of all the 141 sub-studies in the included primary studies, including both core and periphery, there are 31 sub-studies which I was able to identify as having significant prior work. Considering only such prior work that has been found and recorded during the searches, 3 sub-studies replicated such a prior work, and 13 sub-studies otherwise followed up on such a prior work. Table 18 lists all such relationships between primary studies included in this mapping study.

[2] This is the term used, according to my recollection, in the oral presentation of the Tofan et al. (2011) paper.

TABLE 18 Primary studies that replicate or follow up on or are otherwise based on prior work that is itself included in this mapping study

Study	Replicates	Follows up on	Other significant prior work
S18		S29	
S20	S33		
S41			S49, S62
S44		S64	
S56		S55	
S60a		S137, S138	
S60b		S137, S138	
S64			S154
S68		S33	
S79	S141		
S83		S56, S65, S120, S144	
S97			S65, S83, S144
S107		S31	
S122		S19, S20, S33, S68	
S138		S137	
S142			S143
S144		S65	S97, S97
S151		S60, S137, S138	
S158	S125		

6 DISCUSSION

From the results of this mapping study, it is clear that the empirical research on the efficacy of programming language design decisions has a long history, starting from Sime et al. (1973), but it has not been particularly prolific. Before the most recent upsurge of activity, research output had been fairly constant, with an occasional low peak. A significant rise in activity has occurred since 2009. It is notable that this pattern is visible in the data mostly unchanged up to and until removing all but controlled experiments from the data; a major change is seen only when considering solely randomized controlled experiments.

The low level of activity until recent years suggests a rather depressing model of researcher behavior: every once in a while a researcher or a research group comes up with the bright idea that this sort of research would be useful, then conducts a small number of studies, eventually runs out of steam and drops this research area. Both the low number of studies following up on other studies and the general lack of increase in study numbers, excluding the recent years, suggests that the published studies have not been particularly inspiring to other researchers. No paradigm study, in the Kuhn (1996) sense, has emerged to capture the imagination of a generation of researchers; again, disregarding the recent years.

Despite that I have not conducted a formal quality evaluation of the included studies, I think the most plausible explanation is that the studies, not counting the recent couple of years, have simply not been of particularly good quality. This conclusion is reinforced by the fairly scathing critiques of the early studies by Sheil (1981), Hoc (1983), and Détienne (2002). It is also supported by the fact that language designers have generally ignored these studies, as I discussed in Section 2.5. It also accords with the informal impression of the included studies that I have acquired during the conduct of this mapping study.

It is further notable that the most prolific publication forum for the first twenty years, the International Journal of Man–Machine Studies, has all but ceased publishing these studies, despite continuing to be published to this day, albeit with a changed name, the Journal of Human–Computer Studies. Similarly, the premier conference of the programmer behavior research community, the Psychology of Programming Annual Conference (PPIG), has been conspicuously silent with respect to these studies, a fact noted also by Stefik, Hanenberg, et al. (2014). My inclination is to explain these observations by the supposition that the HCI research community has collectively decided that the kinds of studies that my mapping study would consider are not worth the effort. Indeed, Détienne (2002) makes basically this claim: research in the psychology of programming has shifted from a code-centered approach (which is likely relevant to language design) to consider "more removed" (p. 9) topics, related to, for example "specification and design" and teamwork.

Much of what I have just written may apply to the current upsurge of research, or it may not. The absolute numbers are still not very large, indicating that the number of researchers working on the topic may still be fairly low. Whether the current upsurge translates into a sustained growth into a healthy research area or wanes in the next years back to the background levels of the last four decades remains to be seen. The results of this study cannot support any conclusions on that point.

In this study, I have deliberately avoided taking any position as to the conclusions one should make regarding the actual design decisions. For example, I do not offer any analysis on whether static or dynamic typing is better. That task belongs properly to focused systematic literature reviews and is beyond the scope of any mapping study.

The results of this study point to a small number of design decisions that may be ripe for systematic reviewing. The choice between static and dynamic typing, as well as the questions of class inheritance, software transactional memory, conditionals, and program indentation each have at least two randomized controlled experiments and thus it may be possible to synthesize high-quality evidence on them. When considering other core studies, also the question of loops emerges as a potentially viable topic for a systematic review, with its four experiments and one core non-experiment.

It would be too much to expect for any systematic review on these topics to be able to pronounce universally applicable conclusions recommending a single solution to all situations. Instead, as Dybå, Sjøberg, et al. (2012) point out, they are more likely to find, if anything, that each available solution is the best in some context, and perhaps identify which solutions work best in which contexts. That too, would be valuable information.

The same topics that might benefit from systematic reviews are also well enough populated with research that a language designer might actually learn something useful from them.

6.1 Lessons learned

The greatest surprise to me in this mapping study process has been the incredible amount of work it took. My initial estimate was on the order of three or four months; it took three and a half years. The literature searches themselves, producing a total of 2056 recorded publications, not counting duplicates, took almost a year of calendar time (precisely 294 days), though a lot of that is accounted by my teaching duties interfering with this work, and some is accounted by vacations. All in all, on average I seem to have recorded about 7 publications each day (including teaching days and vacations); I am likely to have processed a lot more. The process of going through all the 2056 recorded publications to a final selection decision took almost two years (629 days), meaning an average of 3 decisions every day, including teaching and vacation days. These speeds likely reflect the difficulty of deciding where the line between inclusion and exclusion really lies, based on my definitions of the concepts. A more focused study is likely to be able to attain much higher speeds.

One particular source of trouble was the low general usefulness of abstracts in programming language research. They rarely described what empirical methods, if any, were used to evaluate their work, nor did they usually reveal the results of any such evaluation. As a result, Phase II (based exclusively on online metadata such as abstracts) excluded only about 30 % of the publications. In software enginering, similar problems have been noticed as well, and the use of *structured abstracts* (that is, abstracts with standard explicit subheadings) has been proposed and evaluated with some success (see e. g. Kitchenham, Brereton, Owen, et al. 2008; Budgen, Kitchenham, et al. 2008). I have adopted this practice in the abstract of this study.

I would caution any other research student not to attempt a systematic secondary study alone. An ideal team size is, in my estimate, about six: as recommended by guidelines, each publication should be looked at by at least two researchers independently in each phase of the study, to allow for the estimation of the reliability of decisions; having three teams of two researchers allows significant parallelization of the work. A workable minimum is, I think, three, working together in pairs with a third opinion available for the difficult cases.

In retrospect, the literature search arrangement could have been much more efficient. The problem was that of a bootstrap: I could likely design a more efficient search strategy for this study now, but to get here I had to conduct the inefficient searches. The quasi-gold standard method proposed by Zhang, Ali Babar, and Tell (2011) seems very promising, and I second the recommendation of Kitchenham and Brereton (2013, p. 2068) to incorporate it in future guidelines.

I had initially a lot of trouble with defining the demarcation of evidence. My original plan was to simply take the research method list compiled by Vessey, Ramesh, et al. (2005) as a guide, but it quickly turned out to be unworkable, as they neither define what they mean by the names of the methods nor cite sources for any clear definitions. In the pilot extraction exercise described in Subsection 4.3.1, I and professor Kärkkäinen had significant trouble interpreting the method list. Particular problems for us were the categories DA, data analysis, and LS, laboratory experiment (software).

We debated the question of whether a study that collected existing programs from various sources, ran static analyses and computed metrics on them, and then statistically analyzed the resulting data, could be considered being "based on secondary or existing data" (Vessey, Ramesh, et al. 2005, p. 252) and thus a DA study. Professor Kärkkäinen offered the opinion that all programs are data and thus existing programs are existing data; at the time, I advocated the position that programs in such studies are analogous to human participants and that the metrics derived from them are primary data in each such study. In my later thematic synthesis code book, these studies were allocated the primary method code of CorpusAnalysis or ProgramPairAnalysis, depending on the details of the study.

Similarly, it took some time for us to understand the LS category. Vessey et al. only offered the following comment about it: "We also added [...] Laboratory Experiment (Software) to assist in characterizing computer science/software engineering work." (Vessey, Ramesh, et al. 2005, p. 252). Presumably, it was intended to be an analogy to LH — Laboratory experiment (Human Subjects). A laboratory experiment, according to Alavi and Carlson (1992) (who Vessey et al. cited), "controls for intervening variables". Typically this means assigning some participants to the trial intervention and other participants to a control intervention, but how does one do that when the participants are pieces of software? Eventually we agreed that, for software experiments, control of intervening variables is often implicit as the effect of the control intervention is known *a priori*, and otherwise typically easily instituted by resetting the software before changing interventions (which cannot be done, ethically at least, to humans). This was one of the main motivations for my later definition of an experiment, which differs considerably from the concept of a "true experiment" commonly defined by behavioral researchers; in my taxonomy true experiments would be called randomized controlled experiments. However, in practice, I ended up using non-experimental codes like ProgramRewrite and BenchmarkPrograms for studies of this type.

A problem revealed itself in the Google Scholar search performed on September 7, 2011. It turned out that Google Scholar refuses to display more than one thousand hits. The reported hit count was 2050, and thus the particular search was abandoned under compulsion before the halfway mark was reached. Google (2011) indicates that there is no direct way to overcome this limitation. To try to find the same hits, I conducted the same search with year restrictions, covering together all years, on September 12 and 13, 2011. The combined reported hit count for the piecemeal re-search was 1744, which is 85 % of the reported count of the original abandoned search. A similar tactic for avoiding over-1,000 hits was adopted on subsequent Google Scholar searches.

6.2 Limitations of this study

Every study has limitations; some come from its basic approach, some from its design, and some from problems in its execution. In this section, I will highlight the key limitations of this study.

6.2.1 Conceptual

The concepts of "design decision", "efficacy" and "evidence" are defined in this study in a particular manner, attempting to follow the ordinary meaning of those words but with the goal of giving them a precise content that helps in deciding what studies belong in and what do not belong in this mapping study. Those definitions impose a particular *a priori* model which in some cases is in tension with the model used by the primary studies considered for this mapping study.

6.2.2 Literature search and selection

The quasi-sensitivity of automatic search was, as reported in Subsection 4.1.4, fairly poor. Manual searches fared better, but considering that they have an opportunity to examine all publications in the forums in the quasi-gold standard, the quasi-sensitivity of 85 % is not ideal. The publications in the quasi-gold standard not identified in manual or automatic search were found by snowballing, in some cases over two years after the original searches; it is likely that my understanding of what belongs in the study and what does not had evolved during that time, despite the defined criteria.

The single round of snowballing contributed over half of the publications selected for inclusion, and about 40 % came only from snowballing. Additionally, there are a small number of publications cited by secondary studies (see Subsection 4.1.4) that had not been found or recorded during searches despite being potentially relevant. Since 3 of 30 recorded publications cited by secondary studies had been explicitly rejected, this implies for the 18 not recorded an estimated 90 % survival rate; we may thus assume that 16 of them would have been included. This is a direct consequence of the decision not to do more than one snowballing round. These two observations show clearly that snowball search was stopped before a fixed point was achieved.

The selection validation exercises documented in Subsection 4.2.3 show excellent agreement between myself and my own retest, which is expected due to learning effects, and between myself and one of my supervisors, Ville Tirronen, which is more significant since both of us have studied and taught matters related to programming language research for over a decade. Agreement between myself and another supervisor, Vesa Lappalainen, as well as between Tirronen and Lappalainen, was substantial. My supervisor Tommi Kärkkäinen was a clear outlier, showing only a fair agreement; unfortunately, I inadvertently destroyed the papers on which he recorded his reasoning behind this divergence, thinking that they were not unique copies, and we have been unable to reconstruct them.

All in all, these considerations show that the search and selection of publications for this mapping study have some clear limitations that affect the credibility of the results reported. A counterbalancing consideration is the deliberate biasing of both search and selection toward overinclusion, which necessitated the separation between core and periphery in the thematic model. On the balance, I do believe that most of the relevant literature has been included, and while there are likely some missing publications, they are unlikely to seriously jeopardize my conclusions.

6.2.3 Thematic synthesis

The validation exercise of coding, described in Subsection 4.3.3, suggests that the code book developed was not quite transparent to another researcher but was fairly stable in my own use. The thematic model is supported by the data but there are other possible ways the model could have been framed; other researchers are likely to have created different models. In the main, however, I believe any such differences are likely not to have made a difference in the results obtained, as they are well supported by the data.

7 CONCLUSION

There is clearly some empirical evidence on the efficacy of language design decisions that could inform evidence-based programming language design; however, it is rather sparse. Significant bodies of research seem to exist only of handful of design decisions.

Language designers may find it informative to familiarize themselves with the studies identified in this mapping study at least on the topics of conditional statements, static versus dynamic typing, loops, class inheritance, software transactional memory, and program indentation, concentrating on topics that each designer find of interest to them.

Researchers contemplating the empirical study of programming behavior for the purpose of informing language design might benefit from examining closely the critique of earlier studies offered by Sheil (1981), Hoc (1983), and Détienne (2002). It may be beneficial to reflect on what factors made the earlier research fail both to inspire much further work and to capture the interest of language designers, as well as how the same fate might be avoided for the current upsurge of research on this topic.

Finally, as is traditional in systematic secondary studies and as is amply demonstrated by the results of this study, I note that further primary research is needed on the efficacy of various language design decisions that are relevant for modern languages; particularly, the studies on conditionals are so old as to likely require significant updating to account for current conditions. The same topics I highlighted for language designers above may also benefit from focused systematic reviews.

YHTEENVETO (FINNISH SUMMARY)

Näyttöön perustuvan ohjelmointikielten suunnittelun tueksi sopivan empiirisen tutkimusnäytön laajuus. Järjestelmällinen kirjallisuuskartoitus.

Ohjelmointikieliä on tuhansittain, ja niitä luodaan lisää (ja olemassa olevia kieliä muokataan) jatkuvasti. Tämä luonti- ja kehitystyö perustuu yleensä laatijoiden ja kehittäjien omaan tyylitajuun, henkilökohtaisiin mieltymyksiin sekä teoreettiseen tietämykseen. Empiiristä tutkimustietoa ohjelmointikielten ja niiden muutosten hyödyllisyydestä ei käytetä juuri lainkaan. Ohjelmoinnin psykologian tutkimus on kuitenkin yli neljäkymmentä vuotta vanha tieteenala, ja siitä luulisi olevan hyötyä ohjelmointikielten laatijoille ja kehittäjille.

Tuleville lääkäreille on jo useampi vuosikymmen opetettu näyttöön perustuvan lääketieteen mallia: jos lääkäri ei ole varma, miten tulisi toimia jonkin tietyn potilaan ongelman kanssa, ensiksi hän muotoilee vastattavissa olevan kysymyksen; toiseksi hän etsii tutkimuskirjallisuudesta ja siihen perustuvista toisiolähteistä tutkimusnäyttöä, joka vastaa kyseiseen kysymykseen; kolmanneksi hän arvioi tuon näytön luotettavuuden; neljänneksi hän soveltaa tuon tutkimusnäytön antamaa vastausta potilaansa ongelmaan; ja viidenneksi arvioi omaa suoriutumistaan tässä prosessissa. Tämä lääketieteestä peräisin oleva toimintamalli on sittemmin otettu soveltuvin osin käyttöön myös monilla muilla asiantuntijuuteen perustuvilla aloilla, muiden muassa ohjelmistotekniikassa.

Tämän lisensiaatintyöni lähtökohtana oli näyttöön perustuvan ohjelmointikielten suunnittelun idea. Työn tarkoituksena oli selvittää, kuinka paljon sellaista empiiristä tutkimusnäyttöä on olemassa, josta voisi olla hyötyä ohjelmointikielten suunnittelijoille. Keskityin tarkastelemaan tutkimuksia, jotka pyrkivät vertailemaan kahden tai useamman vaihtoehtoisen suunnitteluratkaisun hyödyllisyyttä ohjelmoijan näkökulmasta. Halusin selvittää lisäksi, mitä tällaisia suunnitteluratkaisuja on tutkittu tällä tavalla, millä eri tavoin hyödyllisyys on ymmärretty tällaisissa tutkimuksissa, sekä mitä tutkimusmenetelmiä tällaisissa tutkimuksissa on käytetty.

Tämä lisensiaatintyöni on kirjallisuuteen perustuva tutkimus, niin sanottu toisiotutkimus, jossa aineistona käytetään ensiötutkimuksia eli tutkimuksia, joissa tutkijat ovat itse välittömästi havainnoineet tutkittavaa ilmiötä. Useimmat järjestelmälliset toisiotutkimukset kuuluvat kahteen pääluokkaan. Järjestelmälliset katsaukset pyrkivät vastaamaan käytännön toiminnan kannalta oleellisiin, hyvin tarkkarajaisiin kysymyksiin. Järjestelmälliset kartoitukset puolestaan pyrkivät hahmottamaan tutkimuskirjallisuuden yleisen tilanteen jollakin tutkimusalalla. Tämä työni on selkeästi kartoitus.

Olen taustoittamisen tarkoituksessa käsitellyt tässä työssäni ohjelmointikielten erilaisia luokitteluja (kielten tasot, sukupolvet ja paradigmat), kielten käsitteellistä rakennetta, tiettyjen suunnitteluratkaisujen historiaa sekä ohjelmointikielten kehitystyön historiaa. Lisäksi olen työssäni suhteellisen laajasti referoinut ohjelmistotekniikan alalla julkaistuja systemaattisten kirjallisuuskartoitusten tutkimusmetodologisia toimintaohjeita. Työni sisältää myös ohjelmointikielen käsitteen analyysiä sekä näytön käsitteen tietoteoreettista pohdintaa.

Itse kartoituksen lähdemateriaalin etsin useita eri hakumenetelmiä käyttäen. Ensiksi selasin läpi eräiden kansainvälisten tutkimuslehtien ja konferenssijulkaisujen kaikki numerot (käyttäen hyväksi tietoverkossa julkaistuja sisällysluetteloja ja abstrakteja). Seuraavaksi tein avainsanahakuja useissa kansainvälisesti tunnetuissa tutkimuskirjallisuustietokannoissa. Lopuksi etsin lisälähteitä kaikkien edellisillä hauilla löytyneiden kartoitukseeni hyväksymieni tutkimusjulkaisuiden lähdeluetteloista sekä eräiden tietokantojen luetteloista näihin julkaisuihin viittaavista julkaisuista; tätä kutsun jatkossa lumipallohauksi.

Hauilla löytyneet julkaisut kävin läpi kolmessa kierroksessa. Ensimmäisellä kierroksella hylkäsin tutkimukseni kannalta ilmiselvästi epäolennaiset julkaisut. Toisella kierroksella hylkäsin ne julkaisut, joiden epäolennaisuudesta olin vakuuttunut. Näillä kahdella kierroksella päätökseni perustuivat tietoverkosta saataviin metatietoihin. Kolmatta kierrosta varten hankin jokaisesta vielä jäljellä olevasta julkaisusta sen koko tekstisisällön, joko paperilla tai sähköisesti. Tällä kierroksella hylkäsin ne, joiden epäolennaisuudesta vakuutuin; loput otin mukaan tähän tutkimukseen. Valintojen oikeellisuuden selvittämiseksi lisensiaatintyöni ohjaajat tekivät kukin pienelle osalle löytyneistä julkaisuista satunnaisotannalla itsenäisen hyväksymis- tai hylkäyspäätöksen. Olimme pääosin samaa mieltä; erimielisyydet ratkaisimme lopullisesti konsensuspäätöksellä.

Mukaan kartoitukseen otin ne ensiö- ja toisiotutkimukset, jotka pyrkivät selvittämään jonkin ohjelmointikielten suunnitteluratkaisun hyödyllisyyden ohjelmoijan näkökulmasta, joista oli saatavilla täydellinen, viimeistään vuonna 2012 julkaistu tutkimusraportti englannin, suomen tai ruotsin kielellä ja jotka esittivät empiiristä tutkimusnäyttöä väitteittensä tueksi.

Selaamalla löytyi 1515 ensimmäisen kierroksen hyväksymää julkaisua, avainsanahauilla löytyi 248 lisää ja lumipallohaulla vielä 293 julkaisua näiden lisäksi. Toisella kierroksella jäljelle jäi 1045 selaamalla löytynyttä, 151 avainsanahauilla löytynyttä ja 223 lumipallohaun löytämää. Lopullisesti kartoitukseen hyväksyttiin 180 tutkimusjulkaisua, jotka raportoivat 137 ensiötutkimusta. Toisiotutkimuksia julkaisuissa raportoitiin 19. Varsinaisessa kartoituksessa olen käsitellyt vain ensiötutkimuksia.

Tein tutkimukseen mukaan otetuista tutkimusjulkaisuista temaattisen synteesin seuraavasti. Ensiksi luin kaikki mukaan otetut julkaisut läpi. Seuraavaksi valitsin jokaisesta suoria lainauksia, jotka liittyivät tutkimukseni aiheeseen. Tämän jälkeen koodasin lainaukset (eli annoin niille kuvaavia avainsanoja). Koodien perusteella etsin aineistosta esille nousevia, tutkimukseni aiheen kannalta merkittäviä teemoja, joista lopulta rakensin temaattisen mallin. Koodauksen oikeellisuuden arvioimiseksi yksi ohjaajistani koodasi muutaman artikkelin uudestaan; ratkaisumme erosivat jonkin verran toisistaan.

Temaattinen mallini jakoi kartoitukseen mukaan ottamani ensiötutkimukset kahteen luokkaan. Reuna-alueeseen kuuluivat tutkimukset, jotka eivät olleet kovin oleellisia kartoitukseni kannalta: ne vain vertailivat kieliä tai kieliluokkia toisiinsa taikka käyttivät yksittäisiä olemassa olevia ohjelmia tai ohjelmointitehtäviä jonkin teknologian käyttökelpoisuuden osoittamiseen. Loput 65 tutkimusta muodostivat ytimen, joka puolestaan jakautui sipulimaisesti useaan kerrokseen käytetyn tutkimusmenetelmän mukaan.

Ydinsipulin uloin kerros koostui tutkimuksista, joissa ei käytetty minkäänlaista koeasetelmaa; tyypillisesti kyse oli määrällisestä havainnoivasta tutkimuksesta taikka laadullisesta tutkimuksesta. Seuraavaksi uloin kerros koostui kokeista eli tutkimuksista, joissa tutkijat ovat pyrkineet vaikuttamaan tutkimustilanteeseen siten, että tästä aiheutuva muutos tulosmittareissa on havaittavissa. Seuraava, toiseksi sisin, kerros koostui kontrolloiduista kokeista eli tutkimuksista, joissa koehenkilöt tai muut tutkimuskohteet on jaettu ryhmiin sen mukaan, mitä tutkimuksessa mukana olevaa suunnitteluratkaisua he käyttävät tai missä järjestyksessä he käyttävät mukana olevia suunnitteluratkaisua. Ydinsipulin sisin kerros eli sydän koostui satunnaistetuista kontrolloiduista kokeista eli kontrolloiduista kokeista, joissa koehenkilöt tai muut tutkimuskohteet on jaettu ryhmiin jollakin satunnaisprosessilla. Sipulin sydämessä oli 22 tutkimusta.

Tutkimusten julkaisuajoista oli havaittavissa mielenkiintoinen ilmiö. Vanhin kartoituksessa mukana ollut julkaisu oli julkaistu 1973 ja uusin vuonna 2012 (koska uudempia en ottanut kartoitukseen mukaan). Aina vuosituhannen vaihteen paikkeille asti tutkimuksia julkaistiin suunnilleen saman verran joka vuosi, mutta määrät nousivat vuosituhannen vaihteen paikkeilla ja uudestaan dramaattisesti vuoden 2008 paikkeilla. Vastaava ilmiö on havaittavissa, joskin heikompana, kaikissa ydinsipulin kerroksissa.

Kartoituksessa havaitsin, että ohjelmointikielten suunnitteluratkaisujen hyödyllisyyttä on tutkittu jonkin verran: kaiken kaikkiaan tutkimuksia löytyi 141 ja satunnaistettuja kontrolloituja kokeita 22. Eniten on tutkittu eri tapoja ilmaista suorituksen haarautumista (11 koetta ytimessä, joista 8 kontrolloituja, joista 3 satunnaistettuja; vanhin tutkimus julkaistu 1973), valintaa staattisen ja dynaamisen tyypityksen välillä (6 tutkimusta ytimessä, joista 5 kontrolloituja kokeita, joista 4 satunnaistettuja; vanhin tutkimus julkaistu 2009), sekä eri tapoja ilmaista silmukkarakenne (5 tutkimusta ytimessä, joista 4 kokeita, joista 3 kontrolloituja ja yksi satunnaistettu; vanhin tutkimus julkaistu 1978). Hyödyllisyyttä on tutkimuksissa tarkasteltu pääasiassa virhealttiuden, ohjelmien ymmärrettävyyden sekä ohjelmointityön työläyden kautta.

Tutkimusmenetelmistä suosituin ytimessä oli (määrällinen) koe, jota käytti 41 tutkimusta. Toiseksi suosituin 11 tutkimuksella oli tutkimusasetelma, jossa olemassa olevia ohjelmia muokattiin käyttämään uutta ohjelmointikielen suunnitteluratkaisua hyväkseen. Kolmanneksi suosituin 8 tutkimuksella oli ohjelmistokorpuksen analyysi. Ytimessä käytettiin lisäksi tapaustutkimusta (2), kyselyä (2) ja ohjelmaparien analysointia (1). Ytimen kokeellisissa tutkimuksissa yleisimmin koehenkilöinä käytettiin ohjelmoijia (35 koetta), jotka tavallisimmin olivat ohjelmoinnin opiskelijoita (29 koetta).

Kartoituksen tuloksista on pääteltävissä varsin masentava kuva tämän kartoituksen alueeseen kuuluvasta tutkimusaktiviteetista. Vaikuttaa siltä, että aina silloin tällöin joku tutkija tai tutkimusryhmä keksii, että tällaiset tutkimukset olisivat hieno juttu, ja tekee niitä sitten muutaman kunnes kyllästyy ja vaihtaa aihetta. Julkaistut tutkimukset eivät vaikuttaisi inspiroineen kovin paljoa jatkotutkimuksia, eikä paradigman perustavia esimerkkitutkimuksia näytä syntyneen. On kuitenkin mahdollista, että viimeisen viiden vuoden aikana lisääntynyt tutkimustoiminta tarkoittaa, että tilanne on muuttunut; mutta koska lukumäärät ovat edelleen pieniä, saattaa tilanne palata jokusen vuoden jälkeen takaisin matalan aktiviteetin tasolle. Valitettavasti kartoitukseni aineistosta ei ole mahdollista päätellä mitään viime vuosien tutkimustoiminnasta.

Lisensiaatintyöni kuluessa tein havainnon, että ohjelmointikielten alan tutkimusartikkeleiden tiivistelmät ovat varsin hyödyttömiä, sillä niissä ei useinkaan kerrota tutkimuksen empiirisen osan metodia eikä sillä saatuja tuloksia. Tähän voisi mahdollisesti saada hyötyä muilla aloilla jo käytössä olevasta rakenteisen tiivistelmän ideasta, jota olen tämänkin työn englanninkielisessä tiivistelmässä (abstract) soveltanut.

Kuten kaikilla tutkimuksilla, tällä lisensiaatintyöllä on rajoitteita, jotka tulee tuloksia tulkittaessa ottaa huomioon. Keskeisin rajoite on, että julkaisujen mukaan ottamisessa ja tutkimusten koodauksessa on voinut sattua virheitä, vaikka niitä on pyritty välttämään ja löytämään. On myös mahdollista, että joitakin asiaan liittyviä tutkimuksia ei ole löytynyt hauissa eikä siksi ole kartoituksessa huomioitu.

Kartoitukseni johtopäätös on, että näyttöön pohjautuvan ohjelmointikielten suunnittelun tueksi on olemassa jonkin verran empiiristä tutkimusnäyttöä, mutta vain muutamaa suunnitteluratkaisua on tutkittu laajemmin. Kielten suunnittelijat saattavat hyötyä kartoituksessa löydettyihin tutkimuksiin tutustumisesta, erityisesti haarautumista, silmukkaa, staattista ja dynaamista tyypitystä, luokkaperintää, tapahtumapohjaista muistia ja sisennystä koskien. Kartoituksen alan kuuluvaa tutkimusta harjoittavien tutkijoiden on syytä tutustua kritiikkiin, jota kirjallisuudessa on esitetty aiempia tutkimuksia vastaan. Lisäksi, kuten järjestelmällisissä toisiotutkimuksissa on tapana, totean, että uusien ensiötutkimusten tekeminen on tarpeen; erityisesti haarautumista koskevat tutkimukset ovat jo iäkkäitä eivätkä ne välttämättä vastaa kovin hyvin nykyoloja. Joistakin aiheista on mahdollisesti myös hyödyllistä laatia järjestelmällisiä katsauksia.

BIBLIOGRAPHY

About Messages and Message Queues 2013. ⟨URL: http://msdn.microsoft.com/en-us/library/windows/desktop/ms644927(v=vs.85).aspx⟩ [visited on 2014-02-12] (cit. on p. 18).

Abrahams, P. W. 1966. A Final Solution to the Dangling **else** of ALGOL 60 and Related Languages. Communications of the ACM 9 (9), 679–682. DOI: 10.1145/365813.365821 (cit. on pp. 27, 28).

ACM s.d.(a). Searching with Words, Phrases, or Plain Language. ⟨URL: http://dl.acm.org/documentation/Types.htm⟩ [visited on 2011-10-19] (cit. on p. 61).

ACM s.d.(b). Zone and Field Searches. ⟨URL: http://dl.acm.org/documentation/Zone.htm⟩ [visited on 2011-10-19] (cit. on p. 61).

Ahmad, A. and Talha, M. 2002. An Empirical Experimentation to Evaluate Effectiveness of Declarative Programming Languages in Software Development Process. In Proc. Software Engineering and Applications (SEA 2002). ⟨URL: http://www.actapress.com/Abstract.aspx?paperId=24494⟩ (cit. on pp. 76, 176).

Aho, A. V., Lam, M. S., Sethi, R., and Ullman, J. D. 2007. Compilers. Principles, Techniques, & Tools. 2nd ed. Boston: Pearson Addison Wesley (cit. on p. 28).

Ahsan, S. N., Ferzund, J., and Wotawa, F. 2009. Are There Language Specific Bug Patterns? Results Obtained from a Case Study Using Mozilla. In Software Engineering Advances, 2009. ICSEA '09. Fourth International Conference on, 210–215. DOI: 10.1109/ICSEA.2009.41 (cit. on pp. 76, 176).

Alavi, M. and Carlson, P. 1992. A Review of MIS Research and Disciplinary Development. Journal of Management Information Systems 8 (4), 45–62 (cit. on pp. 49, 95).

Aldrich, J., Chambers, C., and Notkin, D. 2002. Architectural Reasoning in ArchJava. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374, 334–367. DOI: 10.1007/3-540-47993-7_15 (cit. on pp. 76, 176).

Allen, J. D., Anderson, D., Becker, J., Cook, R., Davis, M., Edberg, P., Everson, M., Freytag, A., Jenkins, J. H., McGowan, R., Moore, L., Muller, E., Phillips, A., Suignard, M., and Whistler, K., eds. The Unicode Standard 2013. Unicode Consortium. ⟨URL: http://www.unicode.org/versions/latest/⟩ (cit. on p. 24).

Allende, E., Callaú, O., Fabry, J., Tanger, É., and Denker, M. 2013. Gradual typing for Smalltalk. Science of Computer Programming. In press. DOI: 10.1016/j.scico.2013.06.006 (cit. on p. 31).

Ambler, A. L., Burnett, M. M., and Zimmerman, B. A. 1992. Operational versus Definitional. A Perspective on Programming Paradigms. Computer 25 (9), 28–43. DOI: 10.1109/2.156380 (cit. on p. 22).

American Standard Code for Information Interchange 1963. Communications of the ACM 6 (8), 422–426. DOI: 10.1145/366707.367524 (cit. on p. 24).

Andreae, C., Coady, Y., Gibbs, C., Noble, J., Vitek, J., and Zhao, T. 2006. Scoped Types and Aspects for Real-Time Java. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067, 124–147. DOI: 10.1007/11785477_7 (cit. on pp. 76, 176).

Arblaster, A. 1982. Human factors in the design and use of computing languages. International Journal of Man-Machine Studies 17 (2), 211–224. DOI: 10.1016/S0020-7373(82)80020-5 (cit. on pp. 36, 76, 180, 181).

Atwood, K. 2008. Homeopathy and Evidence-Based Medicine. Back to the Future Part V. Science-Based Medicine (Feb. 8, 2008). ⟨URL: http://www.sciencebasedmedicine.org/homeopathy-and-evidence-based-medicine-back-to-the-future-part-v/⟩ (cit. on p. 51).

Avison, D., Lau, F., Myers, M., and Nielsen, P. A. 1999. Action Research. Communications of the ACM 42 (1), 94–97. DOI: 10.1145/291469.291479 (cit. on p. 49).

Backus, J. W., Beeber, R. J., Best, S., Goldberg, R., Herrick, H. L., Hughes, R. A., Mitchell, L. B., Nelson, R. A., Nutt, R., Sayre, D., Sheridan, P. B., Stern, H., and Ziller, I. 1956. Programmer's Reference Manual. The FORTRAN Automatic Coding System fort the IBM 704 EDPM. International Business Machines Corporation. New York, Oct. 1956. ⟨URL: http://archive.computerhistory.org/resources/text/Fortran/102649787.05.01.acc.pdf⟩ [visited on 2014-04-15] (cit. on pp. 27, 29).

Backus, J. 1981. The History of FORTRAN I, II, and III. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 25–45. DOI: 10.1145/800025.1198345 (cit. on p. 35).

Badreddin, O., Forward, A., and Lethbridge, T. C. 2012. Model oriented programming: an empirical study of comprehension. In Proceedings of the 2012 Conference of the Center for Advanced Studies on Collaborative Research, 73–86. ⟨URL: http://www.engineering.uottawa.ca/downloads/pdf/Model_Oriented_Programming_An_Empirical_Study_of_Comprehension.pdf⟩ (cit. on pp. 76, 176).

Badreddin, O. and Lethbridge, T. C. 2012. Combining experiments and grounded theory to evaluate a research prototype: Lessons from the umple model-oriented programming technology. In User Evaluation for Software Engineering Researchers (USER), 2012, 1–4. DOI: 10.1109/USER.2012.6226575 (cit. on pp. 76, 176).

Badri, M., Kout, A., and Badri, L. 2012. On the effect of aspect-oriented refactoring on testability of classes: A case study. In Computer Systems and Industrial Informatics (ICCSII), 2012 International Conference on, 1–7. DOI: 10.1109/ICCSII.2012.6454577 (cit. on pp. 76, 176).

Bailey, J., Zhang, C., Budgen, D., Turner, M., and Charters, S. 2007. Search Engine Overlaps. Do they agree or disagree? In Proceedings of the Second International Workshop on Realising Evidence-Based Software Engineering (REBSE 2007). DOI: 10.1109/REBSE.2007.4 (cit. on p. 43).

Baranowski, M. 2002. Current usage of the epicene pronoun in written English. Journal of Sociolinguistics 6 (3), 378–397. DOI: 10.1111/1467-9481.00193 (cit. on p. 6).

Barendregt, H. and Hemerik, K. 1990. Types in Lambda Calculi and Programming Languages. In ESOP'90. 3rd European Symposium on Programming. Ed. by Jones, N. Lecture Notes in Computer Science 432. Springer, 1–35. DOI: 10.1007/3-540-52592-0_53 (cit. on p. 32).

Barnes, F. R. M. and Welch, P. H. 2001. Mobile Data, Dynamic Allocation and Zero Aliasing: an occam Experiment. In Communicating Process Architectures 2001. Ed. by Chalmers, A., Mirmehdi, M., and Muller, H. Concurrent Systems Engineering Series 59. Amsterdam: IOS, 243–264. ⟨URL: http://kar.kent.ac.uk/13552/⟩ (cit. on pp. 76, 176).

Bartsch, M. and Harrison, R. 2008. An exploratory study of the effect of aspect-oriented programming on maintainability. Software Quality Journal 16 (1), 23–44. DOI: 10.1007/s11219-007-9022-7 (cit. on pp. 76, 176).

Bem, D. J. 2011. Feeling the Future. Experimental Evidence for Anomalous Retroactive Influences on Cognition and Affect. Journal of Personality and Social Psychology 100 (3), 407–425. DOI: 10.1037/a0021524 (cit. on p. 51).

Benander, A. C. and Benander, B. A. 1997. C or Pascal as an Introductory CIS Programming Language? An Empirical Study of Student Experience and Performance. The Journal of Computer Information Systems 37 (3), 85–90 (cit. on pp. 76, 176).

Benton, N., Cardelli, L., and Fournet, C. 2004. Modern concurrency abstractions for C#. ACM Transactions on Programming Languages and Systems 26 (5), 769–804. DOI: 10.1145/1018203.1018205 (cit. on pp. 76, 176).

Bergin Jr., T. J. and Gibson Jr., R. G., eds. History of Programming Languages—II 1996. New York: ACM Press (cit. on p. 34).

Bero, L. and Rennie, D. 1995. The Cochrane Collaboration. Preparing, Maintaining, and Disseminating Systematic Reviews of the Effects of Health Care. JAMA 274 (24), 1935–1938. DOI: 10.1001/jama.1995.03530240045039 (cit. on p. 41).

Biermann, A. W., Ballard, B. W., and Sigmon, A. H. 1983. An experimental study of natural language programming. International Journal of Man-Machine Studies 18 (1), 71–87. DOI: 10.1016/S0020-7373(83)80005-4 (cit. on pp. 76, 176).

Bird, A. 1998. Philosophy of Science. UCL Press (cit. on p. 52).

Blackwell, A. F. 2002. What is Programming? In Proceedings of the 14th Annual Workshop of the Psychology of Programming Interest Group (PPIG). ⟨URL: http://www.ppig.org/papers/14th-blackwell.pdf⟩ [visited on 2014-04-08] (cit. on pp. 18, 19).

Blackwell, A. and Green, T. 2003. Notational Systems. The Cognitive Dimensions of Notations Framework. In HCI Models, Theories, and Frameworks. Towards a Multidisciplinary Science. Ed. by Carroll, J. M. Morgan Kaufmann, 103–133 (cit. on p. 37).

Bocchino Jr., R. L., Heumann, S., Honarmand, N., Adve, S. V., Adve, V. S., Welc, A., and Shpeisman, T. 2011. Safe nondeterminism in a deterministic-by-default parallel language. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL), 535–548. DOI: 10.1145/1926385.1926447 (cit. on pp. 76, 176).

Boehm-Davis, D. A. 2002. Empirical Research on Program Comprehension. In Encyclopedia of Software Engineering. John Wiley & Sons, Inc. DOI: 10.1002/0471028959.sof103 (cit. on pp. 76, 180, 181).

Briand, L., Arisholm, E., Counsell, S., Houdek, F., and Thévenod–Fosse, P. 1999. Empirical Studies of Object-Oriented Artifacts, Methods, and Processes: State of the Art and Future Directions. Empirical Software Engineering 4 (4), 387–404. DOI: 10.1023/A:1009825923070 (cit. on pp. 76, 180, 181).

Brooks, F. 1981. APL Session. Transcript of discussant's remarks. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 683–685. DOI: 10.1145/800025.1198425 (cit. on p. 34).

Brown, N. C., Andreazza, A. C., and Young, L. T. 2014. An updated meta-analysis of oxidative stress markers in bipolar disorder. Psychiatry Research. In press. DOI: 10.1016/j.psychres.2014.04.005 (cit. on p. 40).

Budgen, D., Burn, A., Brereton, O. P., Kitchenham, B. A., and Pretorius, R. 2011. Empirical evidence about the UML. A systematic literature review. Software – Practice and Experience 41 (4), 363–392. DOI: 10.1002/spe.1009 (cit. on p. 45).

Budgen, D., Boegh, J., and Mohan, A. 2003. Organising Evidence to support Software Engineering Practice. Report from a Workshop held at STEP 2003, Amsterdam, September 2003. In Proceedings of the Eleventh Annual International Workshop on Software Technology and Engineering Practice (STEP'03). DOI: 10.1109/STEP.2003.23 (cit. on p. 41).

Budgen, D., Kitchenham, B. A., Charters, S. M., Turner, M., Brereton, P., and Linkman, S. G. 2008. Presenting software engineering results using structured abstracts. A randomized experiment. Empirical Software Engineering 13 (4), 435–468. DOI: 10.1007/s10664-008-9075-7 (cit. on p. 94).

Budgen, D., Turner, M., Brereton, P., and Kitchenham, B. 2008. Using Mapping Studies in Software Engineering. In Proc. 20th Annual Workshop of the Psychology of Programming Interest Group (PPIG). ⟨URL: http://ppig.org/papers/20th-budgen.pdf⟩ [visited on 2014-03-13] (cit. on pp. 42, 45).

Burckhardt, S., Leijen, D., Sadowski, C., Yi, J., and Ball, T. 2011. Two for the price of one: a model for parallel and incremental computation. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM, 427–444. DOI: 10.1145/2048066.2048101 (cit. on pp. 76, 176).

Cacho, N., Dantas, F., Garcia, A., and Castor, F. 2009. Exception Flows Made Explicit: An Exploratory Study. In Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on, 43–53. DOI: 10.1109/SBES.2009.11 (cit. on pp. 76, 176).

Campbell, D. T. and Stanley, J. C. 1963. Experimental and Quasi-Experimental Designs for Research. Reprinted from the Handbook of Research on Teaching. Chicago: Rand McNally (cit. on pp. 50, 51).

Campbell-Kelly, M. 1980a. Programming the EDSAC. Early Programming Activity at the University of Cambridge. Annals of the History of Computing 2 (1) (Jan. 1980), 7–36. DOI: 10.1109/MAHC.1980.10009 (cit. on p. 20).

Campbell-Kelly, M. 1980b. Programming the Mark I. Early Programming Activity at the University of Manchester. Annals of the History of Computing 2 (2) (Apr. 1980), 130–168. DOI: 10.1109/MAHC.1980.10018 (cit. on p. 20).

Cardelli, L. 2004. Type Systems. In CRC Handbook of Computer Science and Engineering. Ed. by Tucker, A. B. 2nd ed. CRC. Chap. 97. ⟨URL: http://lucacardelli.name/Papers/TypeSystems.pdf⟩ [visited on 2014-04-22] (cit. on pp. 32, 33).

Cardelli, L. and Wegner, P. 1985. On Understanding Types, Data Abstraction, and Polymorphism. Computing Surveys 17 (4), 471–522. DOI: 10.1145/6041.6042 (cit. on pp. 31–33).

Cartwright, M. 1998. An empirical view of inheritance. Information and Software Technology 40 (14), 795–799. DOI: 10.1016/S0950-5849(98)00105-0 (cit. on pp. 70, 76, 79, 80, 176, 181).

Cartwright, N. and Stegenga, J. 2011. A Theory of Evidence for Evidence-Based Policy. In ed. by Dawid, P., Twining, W., and Vasilaki, M. Proceedings of the British Academy 171. Oxford University Press, 291–322 (cit. on pp. 51, 52).

Castor, F., Cacho, N., Figueiredo, E., Garcia, A., Rubira, C. M. F., Amorim, J. S. de, and Silva, H. O. da 2009. On the modularization and reuse of exception handling with aspects. Software: Practice and Experience 39 (17), 1377–1417. DOI: 10.1002/spe.939 (cit. on pp. 76, 176).

Castor, F., Oliveira, J. P., and Santos, A. L. M. 2011. Software transactional memory vs. locking in a functional language: a controlled experiment. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, & VMIL'11. New York, NY, USA: ACM, 117–122. DOI: 10.1145/2095050.2095071 (cit. on pp. 76, 79, 80, 176).

Ceri, S., Bozzon, A., Brambilla, M., Della Valle, E., Fraternali, P., and Quarteroni, S. 2013. Web Information Retrieval. Berlin: Springer. DOI: 10.1007/978-3-642-39314-3 (cit. on p. 44).

Cesarini, F., Pappalardo, V., and Santoro, C. 2008. A comparative evaluation of imperative and functional implementations of the imap protocol. In Proceedings of the 7th ACM SIGPLAN workshop on ERLANG. ERLANG '08. New York, NY, USA: ACM, 29–40. DOI: 10.1145/1411273.1411279 (cit. on pp. 76, 176).

Chalin, P. and James, P. R. 2007. Non-null References by Default in Java: Alleviating the Nullity Annotation Burden. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609, 227–247. DOI: 10.1007/978-3-540-73589-2_12 (cit. on pp. 76, 85, 176).

Chalmers, I., Hedges, L. V., and Cooper, H. 2002. A Brief History of Research Synthesis. Evaluation & the Health Professions 25 (1), 12–37. DOI: 10.1177/0163278702025001003 (cit. on p. 41).

Champeaux, D. de, Anderson, A., and Feldhousen, E. 1992. Case study of object-oriented software development. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications, 377–391. DOI: 10.1145/141936.141967 (cit. on pp. 76, 176).

Charles, P., Grothoff, C., Saraswat, V., Donawa, C., Kielstra, A., Ebcioglu, K., Praun, C. von, and Sarkar, V. 2005. X10: an object-oriented approach to non-uniform cluster computing. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 519–538. DOI: 10.1145/1094811.1094852 (cit. on pp. 76, 176).

Chen, H.-G. and Vecchio, R. P. 1992. Nested IF-THEN-ELSE constructs in end-user computing: personality and aptitude as predictors of programming ability. International Journal of Man-Machine Studies 36 (6), 843–859. DOI: 10.1016/0020-7373(92)90076-W (cit. on pp. 76, 83, 176).

Chen, L., Ali Babar, M., and Zhang, H. 2010. Towards and Evidence-Based Understanding of Electronic Data Sources. In Proc. 14th International Conference on Evaluation and Assessment in Software Engineering (EASE). ⟨URL: http://ewic.bcs.org/content/ConWebDoc/34796⟩ [visited on 2014-03-18] (cit. on p. 43).

Cherry, J. M. 1986. An experimental evaluation of prefix and postfix notation in command language syntax. International Journal of Man-Machine Studies 24 (4), 365–374. DOI: 10.1016/S0020-7373(86)80052-9 (cit. on pp. 76, 79, 80, 176).

Chilcott, J., Brennan, A., Booth, A., Karnon, J., and Tappenden, P. 2003. The Role of Modelling in Prioritising and Planning Clinical Trials. Health Technology Assessment 7 (23). ⟨URL: http://www.journalslibrary.nihr.ac.uk/hta/volume-7/issue-23⟩ [visited on 2014-03-17] (cit. on p. 45).

Church, A. 1932. A Set of Postulates for the Foundation of Logic. Annals of Mathematics. 2nd ser. 33 (3), 346–366. DOI: 10.2307/1968337 (cit. on p. 32).

Church, A. 1940. A Formulation of the Simple Theory of Types. Journal of Symbolic Logic 5 (2), 56–68. DOI: 10.2307/2266170. ⟨URL: http://www.jstor.org/stable/2266170⟩ (cit. on p. 32).

Church, A. 1941. The Calculi of Lambda-Conversion. Annals of Mathematics Studies 6. Princeton University Press (cit. on p. 32).

Chwistek, L. 1922. Über die Antinomien der Pinzipien der Mathematik. Matematische Zeitschrift 14 (1), 236–243. DOI: 10.1007/BF01215902. ⟨URL: http://resolver.sub.uni-goettingen.de/purl?PPN266833020_0014⟩ (cit. on p. 32).

Chwistek, L. 1925. The Theory of Constructive Logic. (Principles of Logic and Mathematics). Extracted from the «Annales de la Société Polonaise de Mathématique». Cracow: University Press. ⟨URL: http://name.umdl.umich.edu/AAS7985.0001.001⟩ (cit. on p. 32).

COBOL. Report to Conference on Data Systems Languages Including Initial Specifications for a Common Business Oriented Language (COBOL) for Programming Electronic Digital Computers 1960. Department of Defense. Apr. 1960. ⟨URL: http://bitsavers.org/pdf/codasyl/COBOL_Report_Apr60.pdf⟩ [visited on 2014-04-20] (cit. on p. 30).

Coelho, R., Rashid, A., Garcia, A., Ferrari, F., Cacho, N., Kulesza, U., Staa, A. von, and Lucena, C. 2008. Assessing the Impact of Aspects on Exception Flows: An Exploratory Study. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142, 207–234. DOI: 10.1007/978-3-540-70592-5_10 (cit. on pp. 76, 176).

Cohen, J. 1960. A Coefficient of Agreement for Nominal Scales. Educational and Psychological Measurement 20 (1), 37–46. DOI: 10.1177/001316446002000104 (cit. on pp. 46, 67).

Cohen, M., Zhu, H. S., Senem, E. E., and Liu, Y. D. 2012. Energy types. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM, 831–850. DOI: 10.1145/2384616.2384676 (cit. on pp. 76, 85, 176).

Colburn, T. R. 2000. Philosophy and Computer Science. Armonk: Sharpe (cit. on p. 17).

Colquhoun, D. 2011. In Praise of Randomisation. The Importance of Causality in Medicine and its Subversion by Philosophers of Science. In Evidence, Inference and Enquiry. Ed. by Dawid, P., Twining, W., and Vasilaki, M. Proceedings of the British Academy 171. Oxford University Press, 323–343 (cit. on p. 52).

Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science 2013. Dec. 2013. DOI: 10.1145/2534860 (cit. on p. 22).

Computing Curricula 2001. Computer Science Final Report 2001. Dec. 2001. DOI: 10.1145/384274.384275 (cit. on p. 22).

Condit, J., Harren, M., McPeak, S., Necula, G. C., and Weimer, W. 2003. CCured in the real world. In Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation. PLDI '03. New York, NY, USA: ACM, 232–244. DOI: 10.1145/781131.781157 (cit. on pp. 76, 176).

Cook, W. R. 2007. AppleScript. In HOPL III. Proceedings of the third ACM SIGPLAN conference on History of programming languages. New York: ACM. DOI: 10.1145/1238844.1238845 (cit. on pp. 13, 35, 37).

Cowlishaw, M. 1994. The Early History of REXX. IEEE Annals of the History of Computing 16 (4), 15–24. DOI: 10.1109/85.329753 (cit. on p. 35).

Crabtree, A., Rodden, T., Tolmie, P., and Button, G. 2009. Ethnography considered harmful. In CHI'2009 – Digital Life New World. The 27th Annual CHI Conference on Human Factors in Computing Systems. Vol. 2, 879–888. DOI: 10.1145/1518701.1518835 (cit. on p. 49).

Crary, K. and Morrisett, G. 1999. Type Structuere for Low-Level Programming Languages. In Automata, Languages and Programming. 26th International Colloquium, ICALP'99. Ed. by Wiedermann, J., Emde Boas, P. van, and Nielsen, M. Berlin: Springer, 40–54. DOI: 10.1007/3-540-48523-6_4 (cit. on p. 21).

Crocker, D. H. 1982. Standard for the Format of ARPA Internet Text Messages. Request for Comments 822. ⟨URL: http://www.ietf.org/rfc/rfc822.txt⟩ (cit. on p. 149).

Cruzes, D. S. and Dybå, T. 2011a. Recommended Steps for Thematic Synthesis in Software Engineering. In Proceedings of the 2011 Fifth International Symposium on Empirical Software Engineering and Measurement, 275–284. DOI: 10.1109/ESEM.2011.36 (cit. on pp. 48, 68–70).

Cruzes, D. S. and Dybå, T. 2011b. Research synthesis in software engineering. A tertiary study. Information and Software Technology 53 (5), 440–455. DOI: 10.1016/j.infsof.2011.01.004 (cit. on pp. 40, 47).

Cruzes, D., Mendonça, M., Basili, V., Shull, F., and Jino, M. 2007. Automated Information Extraction from Empirical Software Engineering Literature. Is that possible? In ESEM 2007. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, 491–493. DOI: 10.1109/ESEM.2007.62 (cit. on p. 48).

Curtis, B. 1982. A review of human factors research on programming languages and specifications. In Proceedings of the 1982 Conference on Human Factors in Computing Systems. CHI '82. New York, NY, USA: ACM, 212–218. DOI: 10.1145/800049.801782 (cit. on pp. 76, 180, 181).

Dahl, O.-J. and Nygaard, K. 1966. SIMULA. An ALGOL-Based Simulation Language. Communications of the ACM 9 (9), 671–678. DOI: 10.1145/365813.365819 (cit. on p. 30).

Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. 1995. The effect of inheritance on the maintainability of object-oriented software: an empirical study. In Software Maintenance, 1995. Proceedings., International Conference on, 20–29. DOI: 10.1109/ICSM.1995.526524 (cit. on pp. 76, 79, 80, 176).

Daly, J., Brooks, A., Miller, J., Roper, M., and Wood, M. 1996. Evaluating inheritance depth on the maintainability of object-oriented software. Empirical Software Engineering 1 (2), 109–132. DOI: 10.1007/BF00368701 (cit. on pp. 76, 79, 80, 176, 181).

Daly, M. T., Sazawal, V., and Foster, J. S. 2009. Work In Progress: an Empirical Study of Static Typing in Ruby. In Proc. PLATEAU 2009. ⟨URL: http://ecs.victoria.ac.nz/foswiki/pub/Events/PLATEAU/2009Program/plateau09-daly.pdf⟩ (cit. on pp. 76, 79, 80, 88, 176).

Dawes, M., Summerskill, W., Glasziou, P., Cartabellotta, A., Martin, J., Hopayian, K., Porzsolt, F., Burls, A., and Osborne, J. 2005. Sicily statement on evidence-based practice. BMC Medical Education 5. DOI: 10.1186/1472-6920-5-1. [Visited on 2011-09-27] (cit. on p. 38).

Dawid, P., Twining, W., and Vasilaki, M., eds. Evidence, Inference and Enquiry 2011. Proceedings of the British Academy 171. Oxford University Press.

Davis, R. 1977. Generalized procedure calling and content-directed invocation. In Proceedings of the 1977 symposium on Artificial intelligence and programming languages, 45–54. DOI: 10.1145/800228.806931 (cit. on p. 22).

Deligiannis, I. S., Shepperd, M., Webster, S., and Roumeliotis, M. 2002. A Review of Experimental Investigations into Object-Oriented Technology. Empirical Software Engineering 7 (3), 193–231. DOI: 10.1023/A:1016392131540 (cit. on pp. 76, 180, 181).

Demsky, B. and Dash, A. 2008. Bristlecone: A Language for Robust Software Systems. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142, 490–515. DOI: 10.1007/978-3-540-70592-5_21 (cit. on pp. 76, 177).

Denning, P. J. 1989. Editorial: New directions for the *Communications*. Communications of the ACM 32 (2), 164–165. DOI: 10.1145/63342.315917. [Visited on 2011-09-06] (cit. on p. 57).

Dershem, H. L. and Jipping, M. J. 1995. Programming Languages. Structures and Models. Boston: PWS (cit. on pp. 16, 17).

Détienne, F. 2002. Software Design. Cognitive Aspects. Trans. by Bott, F. Practitioner Series. London: Springer (cit. on pp. 13, 18, 36, 92, 93, 98).

Dieste, O., Grimán, A., and Juristo, N. 2009. Developing search strategies for detecting relevant experiments. Empirical Software Engineering 14 (5), 513–539. DOI: 10.1007/s10664-008-9091-7 (cit. on pp. 43, 44).

Dolado, J. J., Harman, M., Otero, M. C., and Hu, L. 2003. An empirical investigation of the influence of a type of side effects on program comprehension. Software Engineering, IEEE Transactions on 29 (7), 665–670. DOI: 10.1109/TSE.2003.1214329 (cit. on pp. 76, 81, 82, 177).

Dolby, J., Hammer, C., Marino, D., Tip, F., Vaziri, M., and Vitek, J. 2012. A data-centric approach to synchronization. ACM Transactions on Programming Languages and Systems 34 (1), 4:1–4:48. DOI: 10.1145/2160910.2160913 (cit. on pp. 76, 85, 177).

Doscher, H. 1990. An Ada case study in cellular telephony testing tools. In Proceedings of the Ada-Europe international conference on Ada : experiences and prospects: experiences and prospects. New York, NY, USA: Cambridge University Press, 24–35. ⟨URL: http://dl.acm.org/citation.cfm?id=103367.103626⟩ (cit. on pp. 76, 177).

Dybå, T., Kitchenham, B. A., and Jørgensen, M. 2005. Evidence-Based Software Engineering for Practitioners. IEEE Software 22 (1), 58–65. DOI: 10.1109/MS.2005.6. [Visited on 2011-09-26] (cit. on pp. 14, 38).

Dybå, T., Sjøberg, D. I. K., and Cruzes, D. S. 2012. What Works for Whom, Where, When, and Why? On the Role of Context in Empirical Software Engineering. In ESEM'12. Proceedings of the ACM–IEEE International Symposium on Empirical Software Engineering and Measurement, 19–28. DOI: 10.1145/2372251.2372256 (cit. on pp. 52, 93).

Dyer, R., Rajan, H., and Cai, Y. 2012. An exploratory study of the design impact of language features for aspect-oriented interfaces. In Proceedings of the 11th annual international conference on Aspect-oriented Software Development. New York, NY, USA: ACM, 143–154. DOI: 10.1145/2162049.2162067 (cit. on pp. 76, 83, 177).

Ebcioğlu, K., Sarkar, V., El-Ghazawi, T., and Urbanic, J. 2006. An experiment in measuring the productivity of three parallel programming languages. In Proceedings of the Third Workshop on Productivity and Performance in High-End Computing (PPHEC-06). ⟨URL: https://upc-bugs.lbl.gov/~phargrov/sc12/PGAS-SC12/content/x10/x10-lang/www.cs.rice.edu/_vs3/PDF/PPHEC2006-final.pdf⟩ (cit. on pp. 76, 177, 181).

Embley, D. W. 1978. Empirical and formal language design applied to a unified control construct for interactive computing. International Journal of Man-Machine Studies 10 (2), 197–216. DOI: 10.1016/S0020-7373(78)80012-1 (cit. on pp. 28, 76, 81, 82, 89, 177, 181).

Embley, D. W. and Hansen, W. J. 1976. The KAIL Selector. A Unified Control Construct. SIGPLAN Notices 11 (1), 22–29. DOI: 10.1145/987324.987327 (cit. on p. 28).

Endrikat, S. and Hanenberg, S. 2011. Is Aspect-Oriented Programming a Rewarding Investment into Future Code Changes? A Socio-technical Study on Development and Maintenance Time. In Program Comprehension (ICPC), 2011 IEEE 19th International Conference on, 51–60. DOI: 10.1109/ICPC.2011.46 (cit. on pp. 76, 177).

Engebretson, A. and Wiedenbeck, S. 2002. Novice comprehension of programs using task-specific and non-task-specific constructs. In Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on, 11–18. DOI: 10.1109/HCC.2002.1046335 (cit. on pp. 76, 79, 80, 177).

Ertl, M. A. 1999. Is Forth Code Compact? A Case Study. In EuroForth'99. ⟨URL: http://www.complang.tuwien.ac.at/papers/ertl99ef.ps.gz⟩ (cit. on pp. 76, 177).

Every-Palmer, S. and Howick, J. 2014. How evidence-based medicine is failing due to biased trials and selective publication. Journal of Evaluation in Clinical Practice. Advance online publication. DOI: 10.1111/jep.12147 (cit. on p. 51).

Evidence-Based Medicine Working Group, the 1992. Evidence-Based Medicine. A New Approach to Teaching the Practice of Medicine. JAMA the Journal of the American Medical Association 268 (17), 2420–2425 (cit. on pp. 14, 38).

Fagan, M. 1991. Soft Typing. An Approach to Type Checking for Dynamically Typed Languages. PhD thesis. Rice University. ⟨URL: http://scholarship.rice.edu/handle/1911/16439⟩ [visited on 2014-02-10] (cit. on p. 17).

Felizardo, K. R., Andery, G. F., Paulovich, F. V., Minghim, R., and Maldonado, J. C. 2012. A visual analysis approach to validate the selection review of primary studies in systematic reviews. Information and Software Technology 54 (10), 1079–1091. DOI: 10.1016/j.infsof.2012.04.003 (cit. on p. 47).

Felizardo, K. R., Salleh, N., Martins, R. M., Mendes, E., MacDonell, S. G., and Maldonado, J. C. 2011. Using Visual Text Mining to Support the Study Selection Activity in Systematic Literature Reviews. In Proceedings of the 2011 Fifth International Symposium on Empirical Software Engineering and Measurement, 77–86. DOI: 10.1109/ESEM.2011.16 (cit. on p. 47).

Felizardo, K. R., Nakagawa, E. Y., Feitosa, D., Minghim, R., and Maldonado, J. C. 2010. An Approach Based on Visual Text Mining to Support Categorization and Classification in the Systematic Mapping. In Proc. 14th International Conference on Evaluation and Assessment in Software Engineering (EASE). ⟨URL: http://ewic.bcs.org/content/ConWebDoc/34783⟩ [visited on 2014-03-20] (cit. on p. 48).

Felizardo, K. R., Riaz, M., Sulayman, M., Mendes, E., MacDonell, S. G., and Maldonado, J. C. 2011. Analyzing the use of graphs to represent the results of Systematic Reviews in Software Engineering. In SBES 2011. 25th Brazilian Symposium on Software Engineering, 174–183. DOI: 10.1109/SBES.2011.9 (cit. on p. 48).

Ferrari, F., Burrows, R., Lemos, O., Garcia, A., Figueiredo, E., Cacho, N., Lopes, F., Temudo, N., Silva, L., Soares, S., Rashid, A., Masiero, P., Batista, T., and Maldonado, J. 2010. An exploratory study of fault-proneness in evolving aspect-oriented programs. In Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1. ICSE '10. New York, NY, USA: ACM, 65–74. DOI: 10.1145/1806799.1806813 (cit. on pp. 76, 84, 85, 177).

Ferrett, L. K. and Offutt, J. 2002. An empirical comparison of modularity of procedural and object-oriented software. In Engineering of Complex Computer Systems, 2002. Proceedings. Eighth IEEE International Conference on, 173–182. DOI: 10.1109/ICECCS.2002.1181510 (cit. on pp. 76, 177).

Fiedler, K. and Krueger, J. I. 2013. Afterthoughts on precognition. No cogent evidence for anomalous influences of consequent events on preceding cognition. Theory & Psychology 23 (3), 323–333. DOI: 10.1177/0959354313485504 (cit. on p. 51).

Figueiredo, E., Cacho, N., Sant'Anna, C., Monteiro, M., Kulesza, U., Garcia, A., Soares, S., Ferrari, F., Khan, S., Filho, F. C., and Dantas, F. 2008. Evolving software product lines with aspects. In Software Engineering, 2008. ICSE '08. ACM/IEEE 30th International Conference on, 261–270. DOI: 10.1145/1368088.1368124 (cit. on pp. 76, 83, 177).

Flanagan, C., Freund, S. N., Lifshin, M., and Qadeer, S. 2008. Types for atomicity: Static checking and inference for Java. ACM Transactions on Programming Languages and Systems 30 (4). DOI: 10.1145/1377492.1377495 (cit. on pp. 76, 177).

Fleiss, J. L. 1971. Measuring Nominal Scale Agreement Among Many Raters. Psychological Bulletin 76 (5), 378–382 (cit. on pp. 46, 67).

Floyd, R. W. 1979. The paradigms of programming. Communications of the ACM 22 (8) (Aug. 1979), 455–460. DOI: 10.1145/359138.359140 (cit. on pp. 22, 24).

FORTRAN IV Language 1963. International Business Machines Corporation. ⟨URL: http://www.fh-jena.de/~kleine/history/languages/C28-6274-1_7090_FORTRANIV.pdf⟩ [visited on 2014-04-15] (cit. on p. 27).

Foster, J. S., Johnson, R., Kodumal, J., and Aiken, A. 2006. Flow-insensitive type qualifiers. ACM Transactions on Programming Languages and Systems 28 (6), 1035–1087. DOI: 10.1145/1186632.1186635 (cit. on pp. 76, 85, 177).

Francis, G. 2012. Too good to be true. Publication bias in two prominent studies from experimental psychology. Psychonomic Bulletin & Review 19 (2), 151–156. DOI: 10.3758/s13423-012-0227-9 (cit. on p. 51).

French, P. 1999. The development of evidence-based nursing. Journal of Advanced Nursing 29 (1), 72–78. DOI: 10.1046/j.1365-2648.1999.00865.x. [Visited on 2011-10-04] (cit. on p. 38).

Friedman, L. W. 1992. From Babbage to Babel and Beyond. A Brief History of Programming Languages. Computer Languages 17 (1), 1–17 (cit. on p. 34).

Furuta, R. and Kemp, P. M. 1979. Experimental evaluation of programming language features: Implications for introductory programming languages. In Proceedings of the tenth SIGCSE technical symposium on Computer science education. New York, NY, USA: ACM, 18–21. DOI: 10.1145/800126.809544 (cit. on pp. 76, 180, 181).

Fyfe, R. 1997a. An Empirical Study of C++ Programs. BSc thesis. ⟨URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.5633&rep=rep1&type=pdf⟩ (cit. on pp. 76, 180, 181).

Fyfe, R. 1997b. An Empirical Study on C++ Programs Project Literature Review. Student project report. ⟨URL: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.142.6326&rep=rep1&type=pdf⟩ (cit. on pp. 76, 180, 181).

Fähndrich, M. and Leino, K. R. M. 2003. Declaring and checking non-null types in an object-oriented language. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications, 302–312. DOI: 10.1145/949305.949332 (cit. on pp. 76, 177).

Gabbrielli, M. and Martini, S. 2010. Programming Languages. Principles and Paradigms. London: Springer. DOI: 10.1007/978-1-84882-914-5 (cit. on pp. 17, 26).

Gannon, J. D. 1977. An experimental evaluation of data type conventions. Communications of the ACM 20 (8), 584–595. DOI: 10.1145/359763.359800 (cit. on pp. 76, 81, 82, 89, 177, 181).

Gannon, J. D. and Horning, J. J. 1975a. Language design for programming reliability. Software Engineering, IEEE Transactions on SE-1 (2), 179–191. DOI: 10.1109/TSE.1975.6312838 (cit. on pp. 76, 177, 181).

Gannon, J. D. and Horning, J. J. 1975b. The impact of language design on the production of reliable software. In Proceedings of the international conference on Reliable software. New York, NY, USA: ACM, 10–22. DOI: 10.1145/800027.808420 (cit. on pp. 76, 177, 181).

Gannon, J. D. 1976. An experiment for the evaluation of language features. International Journal of Man-Machine Studies 8 (1), 61–73. DOI: 10.1016/S0020-7373(76)80010-7 (cit. on pp. 76, 177, 181).

García-Borgoñón, L., Barcelona, M. A., García-García, J. A., Alba, M., and Escalona, M. J. 2014. Software process modeling languages. A systematic literature review. Information and Software Technology 56 (2), 103–116. DOI: 10.1016/j.infsof.2013.10.001 (cit. on p. 40).

Gerakios, P., Biboudis, A., and Smaragdakis, Y. 2013. Forsaking Inheritance. Supercharged Delegation in DelphJ. In OOPSLA'13. The Proceedings of the 2013 International Conference on Object Oriented Programming, Systems, Languages & Applications, 233–251. DOI: 10.1145/2509136.2509535 (cit. on p. 13).

Gerakios, P., Papaspyrou, N., and Sagonas, K. 2014. Static safety guarantees for a low-level multithreaded language with regions. Science of Computer Programming 80, 223–263. DOI: 10.1016/j.scico.2013.06.005 (cit. on p. 26).

Gettys, J., Scheifler, R. W., Adams, C., Joloboff, V., Hiura, H., McMahon, B., Newman, R., Tabayoyon, A., Widener, G., and Yamada, S. 2002. Xlib. C Language X Interface. X Consortium Standard. ⟨URL: http://www.x.org/releases/X11R7.7/doc/libX11/libX11/libX11.html⟩ [visited on 2014-02-12] (cit. on p. 18).

Gil, J. and Lenz, K. 2010. The Use of Overloading in Java Programs. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183, 529–551. DOI: 10.1007/978-3-642-14107-2_25 (cit. on pp. 76, 85, 177).

Gil, J. and Shragai, T. 2009. Are We Ready for a Safer Construction Environment? In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653, 495–519. DOI: 10.1007/978-3-642-03013-0_23 (cit. on pp. 76, 85, 177).

Gilmore, D. J. and Green, T. R. G. 1984. Comprehension and recall of miniature programs. International Journal of Man-Machine Studies 21 (1), 31–48. DOI: 10.1016/S0020-7373(84)80037-1 (cit. on pp. 76, 79, 80, 177).

Giloi, W. K. 1997. Konrad Zuse's Plankalkül. The First High-Level, "non von Neumann" Programming Language. IEEE Annals of the History of Computing 19 (2), 17–24. DOI: 10.1109/85.586068 (cit. on p. 34).

Girard, J.-Y. 1971. Une extension de l'interpretation de Gödel a l'analyse, et son application a l'elimination des coupures dans l'analyse et la theorie des types. In Proceedings of the Second Scandinavian Logic Symposium. Ed. by Fenstad, J. E. Studies in logic and the foundations of mathematics 63. Amsterdam: North-Holland, 63–92 (cit. on p. 33).

Glaser, B. G. and Strauss, A. L. 1967. The Discovery of Grounded Theory. Strategies for Qualitative Research. Chicago: Aldine (cit. on p. 49).

Glass, G. V. 1976. Primary, Secondary and Meta-Analysis of Research. Educational Researcher 5 (10), 3–8. ⟨URL: http://www.jstor.org/stable/1174772⟩ (cit. on p. 40).

Glass, R. L., Vessey, I., and Ramesh, V. 2002. Research in software engineering. An analysis of the literature. Information and Software Technology 44, 491–506 (cit. on p. 59).

Godfrey-Smith, P. 2003. Theory and reality. An introduction to the philosophy of science. Chicago: University of Chicago Press (cit. on pp. 52, 53).

Goldberg, A. and Robson, D. 1983. SMALLTALK'80. The language and its implementation. Reading, Massachusetts: Addison–Wesley. ⟨URL: http://dl.acm.org/citation.cfm?id=273⟩ (cit. on p. 31).

Goldstein, I. and Sussman, G. J. 1974. Some Projects in Automatic Programming. Working Paper 67. Massachusetts Institute of Technology Artificial Intelligence Laboratory, Apr. 1974. ⟨URL: https://dspace.mit.edu/handle/1721.1/41102⟩ [visited on 2014-04-11] (cit. on p. 22).

Google 2011. Google Scholar Help. ⟨URL: http://scholar.google.com/intl/en/scholar/help.html⟩ [visited on 2011-09-09] (cit. on p. 95).

Gosling, J., Joy, B., Steele, G., Bracha, G., and Buckley, A. 2014. The Java® Language Specification. Java SE 8 Edition. Oracle. ⟨URL: http://docs.oracle.com/javase/specs/jls/se8/jls8.pdf⟩ [visited on 2014-04-17] (cit. on pp. 17, 29, 157).

Gougen, J. A., Thatcher, J. W., Wagner, E. G., and Wright, J. B. 1977. Initial Algebra Semantics and Continuous Algebras. Journal of the Association for Computing Machinery 24 (1), 68–95. DOI: 10.1145/321992.321997 (cit. on p. 25).

Graunke, P., Krishnamurthi, S., Van Der Hoeven, S., and Felleisen, M. 2001. Programming the Web with High-Level Programming Languages. In Programming Languages and Systems. 10th European Symposium on Programming, ESOP 2001, Held as Part of the Joint Europaean Conferences on Theory and Practice of Software, ETAPS 2001. Ed. by Sands, D. Lecture Notes in Computer Science 2028. Berlin: Springer, 122–136. DOI: 10.1007/3-540-45309-1_9 (cit. on p. 20).

Gray, G. T. and Smith, R. Q. 2004. Sperry Rand's First-Generation Computers, 1955–1960. Hardware and Software. IEEE Annals of the History of Computing 26 (4), 20–34. DOI: 10.1109/MAHC.2004.34 (cit. on p. 34).

Green, J., Shapiro, R. M., Helt Jr., F. R., Franciotti, R. G., and Theil, E. H. 1959. Remarks on ALGOL and Symbol Manipulation. Communications of the ACM 2 (9), 25–27. DOI: 10.1145/368424.368438 (cit. on p. 27).

Green, T. R. G. 1977. Conditional program statements and their comprehensibility to professional programmers. Journal of Occupational Psychology 50 (2), 93–109. DOI: 10.1111/j.2044-8325.1977.tb00363.x (cit. on pp. 76, 83, 177, 181).

Green, T. R. G. 1980. Ifs and thens: Is nesting just for the birds? Software: Practice and Experience 10 (5), 373–381. DOI: 10.1002/spe.4380100505 (cit. on pp. 76, 180, 181).

Green, T. R. G. 1989. Cognitive Dimensions of Notations. In People and Computers V. Ed. by Sutcliffe, A. and Macaulay, L. Cambridge University Press, 443–460. ⟨URL: https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/papers/Green1989.pdf⟩ [visited on 2014-04-26] (cit. on p. 36).

Greenwood, P., Bartolomei, T., Figueiredo, E., Dosea, M., Garcia, A., Cacho, N., Sant'Anna, C., Soares, S., Borba, P., Kulesza, U., and Rashid, A. 2007. On the Impact of Aspectual Decompositions on Design Stability: An Empirical Study. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609, 176–200. DOI: 10.1007/978-3-540-73589-2_9 (cit. on pp. 76, 177).

Griffiths, M. 1975. Relationship Between Definition and Implementation of a Language. In Software Engineering. An Advanced Course. Ed. by Bauer, F. L. Lecture Notes in Computer Science 30. Berlin: Springer, 76–110. DOI: 10.1007/3-540-07168-7_75 (cit. on p. 25).

Guba, E. G. and Lincoln, Y. S. 1994. Competing Paradigms in Qualitative Research. In Handbook of Qualitative Research. Ed. by Denzin, N. K. and Lincoln, Y. S. Thousand Oaks: SAGE, 105–117 (cit. on p. 52).

Guyatt, G. H. 1991. Evidence-Based Medicine [Editorial]. Annals of Internal Medicine 114 (ACP Journal Club supplement 2), A–16 (cit. on pp. 14, 38).

Halpern, J. Y., Meyer, A. R., and Trakhtenbrot, B. A. 1984. The Semantics of Local Storage, or What Makes the Free-List Free. Preliminary report. In POPL'84. Proceedings of the 11th ACM SIGACT–SIGPLAN Symposium on Principles of Programming Languages, 245–257. DOI: 10.1145/800017.800536 (cit. on p. 17).

Halverson R., J. 1993. An empirical investigation comparing IF-THEN rules and decision tables for programming rule-based expert systems. In System Sciences, 1993, Proceeding of the Twenty-Sixth Hawaii International Conference on. Vol. iii, 316–323 vol.3. DOI: 10.1109/HICSS.1993.284327 (cit. on pp. 76, 79, 80, 88, 177).

Hammerstrøm, K. T. and Bjørndal, A. 2011. If there are no randomized controlled trials, do we always need more research? Cochrane Database of Systematic Reviews (Mar. 14, 2011). ⟨URL: http://www.thecochranelibrary.com/details/editorial/1034087/If-there-are-no-randomised-controlled-trials-do-we-always-need-more-research.html⟩ (cit. on p. 51).

Hanenberg, S., Kleinschmager, S., and Josupeit-Walter, M. 2009. Does aspect-oriented programming increase the development speed for crosscutting code? An empirical study. In Third international symposium on Empirical Software Engineering and Measurement ESEM 2009, 156–167. DOI: 10.1109/ESEM.2009.5316028 (cit. on pp. 76, 177).

Hanenberg, S. 2009. What is the Impact of Type Systems on Programming Time? First Empirical Results. In Proc. PLATEAU 2009. ⟨URL: http://ecs.victoria.ac.nz/foswiki/pub/Events/PLATEAU/2009Program/plateau09-hanenberg.pdf⟩ (cit. on pp. 76, 81, 82, 177).

Hanenberg, S. 2010a. An experiment about static and dynamic type systems: doubts about the positive impact of static type systems on development time events. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications, 22–35. DOI: 10.1145/1869459.1869462 (cit. on pp. 62, 76, 81, 82, 177).

Hanenberg, S. 2010b. Doubts about the Positive Impact of Static Type Systems on Programming Tasks in Single Developer Projects - An Empirical Study. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183, 300–303. DOI: 10.1007/978-3-642-14107-2_14 (cit. on pp. 62, 76, 81, 82, 177).

Hanenberg, S. 2010c. Faith, hope, and love. An essay on software science's neglect of human factors. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. Reno/Tahoe, Nevada, USA: ACM, 933–946. ISBN: 978-1-4503-0203-6. DOI: 10.1145/1869459.1869536 (cit. on pp. 13, 37).

Hanenberg, S., Kleinschmager, S., Robbes, R., Tanter, É., and Stefik, A. 2013. An Empirical Study on the impact of Static Typing on Software Maintainability. Empirical Software Engineering. DOI: 10.1007/s10664-013-9289-1 (cit. on p. 31).

Harel, D. and Rumpe, B. 2004. Meaningful Modeling. What's the Semantics of "Semantics"? Computer 37 (10), 64–72. DOI: 10.1109/MC.2004.172 (cit. on p. 26).

Harel, E. C. and McLean, E. R. 1985. The Effects of Using a Nonprocedural Computer Language on Programmer Productivity. MIS Quarterly 9 (2), 109–120. ⟨URL: http://www.jstor.org/stable/249112⟩ (cit. on pp. 76, 177).

Harper, R. 2014. Practical Foundations for Programming Languages. Version 1.43. ⟨URL: http://www.cs.cmu.edu/~rwh/plbook/book.pdf⟩ [visited on 2014-04-22] (cit. on pp. 26, 31).

Harrison, R., Counsell, S., and Nithi, R. 2000. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. Journal of Systems and Software 52 (2–3), 173–179. DOI: 10.1016/S0164-1212(99)00144-2 (cit. on pp. 76, 79, 80, 178, 181).

Harrison, R., Smaraweera, L. G., Dobie, M. R., and Lewis, P. H. 1996. Comparing programming paradigms: an evaluation of functional and object-oriented programs. Software Engineering Journal 11 (4), 247–254. ⟨URL: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=511273&tag=1⟩ (cit. on pp. 76, 177, 181).

Henry, S. M. and Humphrey, M. 1990. A controlled experiment to evaluate maintainability of object-oriented software. In Software Maintenance, 1990., Proceedings., Conference on, 258–265. DOI: 10.1109/ICSM.1990.131370 (cit. on pp. 76, 178, 181).

Henry, S. and Humphrey, M. 1993. Object-oriented vs procedural programming-languages: effectiveness in program maintenance. Journal of Object-Oriented Programming 6 (3), 41–49 (cit. on pp. 76, 178).

Henry, S. M. and Humphrey, M. C. 1988. Comparison of an Object-Oriented Programming Language to a Procedural Programming Language for Effectiveness in Program Maintenance. Technical Report TR-88-49. Computer Science, Virginia Polytechnic Institute and State University. ⟨URL: http://eprints.cs.vt.edu/archive/00000133/⟩ (cit. on pp. 76, 178).

Hertz, M. and Berger, E. D. 2005. Quantifying the performance of garbage collection vs. explicit memory management. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 313–326. DOI: 10.1145/1094811.1094836 (cit. on pp. 76, 178).

Hicks, M., Morrisett, G., Grossman, D., and Jim, T. 2004. Experience with safe manual memory-management in cyclone. In Proceedings of the 4th international symposium on Memory management. New York, NY, USA: ACM, 73–84. DOI: 10.1145/1029873.1029883 (cit. on pp. 76, 178).

Higgins, J. P. and Green, S., eds. Cochrane Handbook for Systematic Reviews of Interventions 2011. Version 5.1.0. The Cochrane Collaboration. ⟨URL: http://handbook.cochrane.org/⟩ (cit. on pp. 42, 43).

Hilbert, D. and Ackermann, W. 1928. Grundzüge der Theoretischen Logik. Die Grundlehren der Matematischen Wissenschaften XXVII. Berlin: Springer (cit. on p. 32).

Hindley, R. 1969. The Principal Type-Scheme of an Object in Combinatory Logic. Transactions of the American Mathematical Society 146 (Dec. 1969), 29–60. DOI: 10.2307/1995158 (cit. on p. 33).

Hitz, M. and Hudec, M. 1995. Modula-2 versus C++ as a first programming language—some empirical results. SIGCSE Bulletin 27 (1), 317–321. DOI: 10.1145/199691.199838 (cit. on pp. 76, 178).

Hoare, C. A. R. 1965. Record Handling. ALGOL Bulletin (21) (Nov. 1965), 39–69. ⟨URL: http://dl.acm.org/citation.cfm?id=1061041⟩ (cit. on p. 30).

Hoare, C. A. R. 1966. Further Thoughts on Record Handling AB21.3.6. ALGOL Bulletin (23) (May 1966), 5–11. ⟨URL: http://dl.acm.org/citation.cfm?id=1061069⟩ (cit. on p. 30).

Hoare, C. A. R. 1989. Hints on programming-language design. In Essays in Computing Science. Ed. by Jones, C. B. Prentice Hall, 193–216 (cit. on p. 33).

Hoc, J.-M. 1983. Psychological study of programming activity: a review. Technology and Science of Informatics 1 (5). ⟨URL: http://jeanmichelhoc.free.fr/pdf/Hoc%201983a.pdf⟩ (cit. on pp. 36, 76, 92, 98, 180–182).

Hoc, J.-M., Green, T. R. G., Samurçay, R., and Gilmore, D. J., eds. Psychology of Programming 1990. London: Academic (cit. on pp. 13, 36).

Hochstein, L. and Basili, V. R. 2006. An empirical study to compare two parallel programming models. In Proceedings of the eighteenth annual ACM symposium on Parallelism in algorithms and architectures. SPAA '06. New York, NY, USA: ACM, 114–114. DOI: 10.1145/1148109.1148127 (cit. on pp. 76, 81, 82, 178).

Hochstein, L., Basili, V. R., Vishkin, U., and Gilbert, J. 2008. A pilot study to compare programming effort for two parallel programming models. Journal of Systems and Software 81 (11), 1920–1930. DOI: 10.1016/j.jss.2007.12.798 (cit. on pp. 76, 81, 82, 178, 181).

Hoffman, K. and Eugster, P. 2008. Towards reusable components with aspects: an empirical study on modularity and obliviousness. In Proceedings of the 30th international conference on Software engineering. ICSE '08. New York, NY, USA: ACM, 91–100. DOI: 10.1145/1368088.1368102 (cit. on pp. 76, 85, 178).

Holmevik, J. R. 1994. Compiling SIMULA. A Historical Study of Technological Genesis. IEEE Annals of the History of Computing 16 (4), 25–37. DOI: 10.1109/85.329756 (cit. on p. 34).

Hopcroft, J. E., Motwani, R., and Ullman, J. D. 2007. Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson Addison Wesley (cit. on p. 17).

HOPL III. Proceedings of the third ACM SIGPLAN conference on History of programming languages 2007. New York: ACM (cit. on p. 34).

Howick, J., Chalmers, I., Glasziou, P., Greenhalgh, T., Heneghan, C., Liberati, A., Moschetti, I., Phillips, B., Thornton, H., Goddard, O., and Hodginkson, M. 2011. The Oxford 2011 Levels of Evidence. ⟨URL: http://www.cebm.net/index.aspx?o=5653⟩ [visited on 2014-04-26] (cit. on p. 51).

Howson, C. and Urbach, P. 2006. Scientific Reasoning. The Bayesian Approach. 3rd ed. Chicago: Open Court (cit. on p. 53).

Hu, R., Kouzapas, D., Pernet, O., Yoshida, N., and Honda, K. 2010. Type-Safe Eventful Sessions in Java. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183, 329–353. DOI: 10.1007/978-3-642-14107-2_16 (cit. on pp. 76, 178).

Huang, S. S. and Smaragdakis, Y. 2011. Morphing: Structurally shaping a class by reflecting on others. ACM Transactions on Programming Languages and Systems 33 (2), 6:1–6:44. DOI: 10.1145/1890028.1890029 (cit. on pp. 76, 178).

Hudak, P., Hughes, J., Peyton Jones, S., and Wadler, P. 2007. A History of Haskell. Being Lazy With Class. In Proceedings of the Third ACM SIGPLAN History of Programming Languages Conference. HOPL–III. DOI: 10.1145/1238844.1238856 (cit. on p. 35).

Hudak, P. and Jones, M. P. 1994. Haskell vs. Ada vs. C++ vs. Awk vs. ...: An Experiment in Software Prototyping Productivity. Tech. rep. Yale University. ⟨URL: http://www.cs.yale.edu/publications/techreports/tr1049.pdf⟩ (cit. on pp. 76, 178).

Hughes, J. 1989. Why Functional Programming Matters. The Computer Journal 32 (2), 98–107. DOI: 10.1093/comjnl/32.2.98 (cit. on p. 23).

IEEE Standard Glossary of Software Engineering Terminology 1990. IEEE Std 610.12-1990. DOI: 10.1109/IEEESTD.1990.101064 (cit. on pp. 16, 20, 21).

Igarashi, A., Pierce, B. C., and Wadler, P. 2001. Featherweight Java. A Minimal Core Calculus for Java and GJ. ACM Transactions on Programming Languages and Systems 23 (3), 396–450. DOI: 10.1145/503502.503505 (cit. on p. 17).

Imtiaz, S., Bano, M., Ikram, N., and Niazi, M. 2013. A Teriary Study. Experiences of Conducting Systematic Literature Reviews in Software Engineering. In EASE 2013. Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering, 177–182. DOI: 10.1145/2460999.2461025 (cit. on p. 42).

Information Technology – Programming Languages – C# 2006. ISO/IEC 23270. ⟨URL: http://standards.iso.org/ittf/PubliclyAvailableStandards/c042926_ISO_IEC_23270_2006(E).zip⟩ [visited on 2014-04-17] (cit. on pp. 29, 157).

Information Technology – Programming Languages – C 2011. INCITS/ISO/IEC 9899 (cit. on pp. 29, 157).

Ioannidis, J. P. A. 2005. Why Most Published Research Findings are False. PLoS Medicine 2 (8), a124. DOI: 10.1371/journal.pmed.0020124 (cit. on pp. 14, 51).

Ioannidis, J. P. A. 2008. Why Most Discovered True Associations Are Inflated. Epidemiology 19 (5), 640–648. DOI: 10.1097/EDE.0b013e31818131e7 (cit. on pp. 14, 51).

Irvine, A. D. and Deutsch, H. 2013. Russell's Paradox. In Stanford Encyclopedia of Philosophy. Ed. by Zalta, E. N. Winter 2013. ⟨URL: http://plato.stanford.edu/archives/win2013/entries/russell-paradox/⟩ (cit. on p. 31).

Iselin, E. R. 1988. Conditional statements, looping constructs, and program comprehension: an experimental study. International Journal of Man-Machine Studies 28 (1), 45–66. DOI: 10.1016/S0020-7373(88)80052-X (cit. on pp. 70, 76, 79, 80, 178).

Jalali, S. and Wohlin, C. 2012. Systematic Literature Studies. Database Searches vs. Backward Snowballing. In ESEM'12. Proceedings of the ACM–IEEE International Symposium on Empirical Software Engineering and Measurement, 29–38. DOI: 10.1145/2372251.2372257 (cit. on p. 45).

Jeffrey, R. 2004. Subjective Probability. The Real Thing. Cambridge University Press (cit. on p. 53).

Jim, T., Morrisett, G., Grossman, D., Hicks, M., Cheney, J., and Wang, Y. 2002. Cyclone: A safe dialect of C. In USENIX Annual Technical Conference. Vol. 90. ⟨URL: http://static.usenix.org/event/usenix02/jim.html⟩ (cit. on pp. 76, 178).

Johnson, R. A. 2002. Object-oriented systems development: A review of empirical research. Communications of the Association for Information Systems 8 (1). ⟨URL: http://aisel.aisnet.org/cais/vol8/iss1/4/⟩ (cit. on pp. 76, 180, 181).

Kaijanaho, A.-J. 2010. Ohjelmointikielten periaatteet. Luentomoniste 16. Jyväskylän yliopisto, tietotekniikan laitos. ⟨URL: http://users.jyu.fi/~antkaij/opetus/okp/2012/okp-moniste.pdf⟩ (cit. on p. 17).

Kamareddine, F., Laan, T., and Nederpelt, R. 2002. Types in Logic and Mathematics before 1940. The Bulletin of Symbolic Logic 8 (2), 185–245. DOI: 10.2307/2693964. ⟨URL: http://www.jstor.org/stable/2693964⟩ (cit. on p. 32).

Katz, J. H. and McGee, W. C. 1963. An Experiment in Non-Procedural Programming. In 1963 Fall Joint Computer Conference. AFIPS Conference Proceedings 24. Spartan. DOI: 10.1145/1463822.1463824 (cit. on p. 23).

Kaupe Jr., A. F. 1963. A Note on the Dangling **else** in ALGOL 60. Communications of the ACM 6 (8), 460–462. DOI: 10.1145/366707.367585 (cit. on p. 27).

Kernighan, B. W. and Ritchie, D. M. 1978. The C programming language. Prentice-Hall (cit. on p. 29).

Kernighan, B. W. and Ritchie, D. M. 1988. The C programming language. 2nd ed. Prentice Hall (cit. on p. 29).

Kesler, T. E., Uram, R. B., Magareh-Abed, F., Fritzsche, A., Amport, C., and Dunsmore, H. E. 1984. The effect of indentation on program comprehension. International Journal of Man-Machine Studies 21 (5), 415–428. DOI: 10.1016/S0020-7373(84)80068-1 (cit. on pp. 76, 79, 80, 178).

Kiczales, G., Hilsdale, E., Hugunin, J., Kersten, M., Palm, J., and Griswold, W. G. 2001. An Overview of AspectJ. In ECOOP 2001 – Object-Oriented Programming. 15th European Conference. Ed. by Knudsen, J. L. Berlin. Lecture Notes in Computer Science 2072. Springer, 327–353. DOI: 10.1007/3-540-45337-7_18 (cit. on p. 23).

Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J.-M., and Irwin, J. 1997. Aspect-Oriented Programming. In ECOOP'97 – Object-Oriented Programming. 11th European Conference. Ed. by Akşit, M. and Matsuoka, S. Lecture Notes in Computer Science 1241. Berlin: Springer, 220–242. DOI: 10.1007/BFb0053381 (cit. on p. 23).

Kilpatrick, S., Dreyer, D., Peyton Jones, S., and Marlow, S. 2014. Backpack. Retrofitting Haskell with Interfaces. In POPL'14. Proceedings of the 41st Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming languages, 19–31. DOI: 10.1145/2535838.2535884 (cit. on p. 13).

Kinnersley, B. 2001. The Language List. Collected Information On About 2500 Computer Languages, Past and Present. ⟨URL: http://people.ku.edu/~nkinners/LangList/Extras/langlist.htm⟩ [visited on 2014-02-21] (cit. on p. 13).

Kitchenham, B. A., Brereton, O. P., Owen, S., Butcher, J., and Jeffries, C. 2008. Length and readability of structured software engineering abstracts. IET Software 2 (1), 37–45. DOI: 10.1049/iet-sen:20070044 (cit. on p. 94).

Kitchenham, B. 2004a. Procedures for Performing Systematic Reviews. Joint Technical Report Keele TR/SE-0401, NICTA 0400011T.1. An early version of Kitchenham and Charters (2007). Keele University and National ICT Australia. ⟨URL: http://www.scm.keele.ac.uk/ease/sreview.doc⟩ [visited on 2014-05-05] (cit. on p. 41).

Kitchenham, B. 2010. What's up with software metrics? A preliminary mapping study. Journal of Systems and Software 83, 37–51. DOI: 10.1016/j.jss.2009.06.041 (cit. on p. 6).

Kitchenham, B. A. 2004b. Systematic Reviews. In Proceedings of the 10th International Symposium on Software Metrics, xii. DOI: 10.1109/METRIC.2004.1357885 (cit. on p. 41).

Kitchenham, B. A., Brereton, P., Turner, M., Niazi, M. K., Linkman, S., Pretorius, R., and Budgen, D. 2010. Refining the Systematic Literature Review Process. Two participant-observer case studies. Empirical Software Engineering 15 (6), 618–653. DOI: 10.1007/s10664-010-9134-8 (cit. on p. 45).

Kitchenham, B. A., Budgen, D., and Brereton, O. P. 2011. Using mapping studies as the basis for further research. A participant-observer case study. Information and Software Technology 53 (6), 638–651. DOI: 10.1016/j.infsof.2010.12.011 (cit. on pp. 40, 42, 43, 45).

Kitchenham, B. A., Dybå, T., and Jørgensen, M. 2004. Evidence-Based Software Engineering. In Proceedings of the 26th International Conference on Software Engineering (ICSE'04). DOI: 10.1109/ICSE.2004.1317449. [Visited on 2011-09-26] (cit. on pp. 14, 38).

Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. 2009. Systematic literature reviews in software engineering. A systematic literature review. Information and Software Technology 51 (1), 7–15. DOI: 10.1016/j.infsof.2008.09.009 (cit. on p. 40).

Kitchenham, B. and Brereton, P. 2013. A systematic review of systematic review process research in software engineering. Information and Software Technology 55 (12), 2049–2075. DOI: 10.1016/j.infsof.2013.07.010 (cit. on pp. 41–43, 45, 47, 94).

Kitchenham, B., Brereton, P., and Budgen, D. 2012. Mapping study completeness and reliability. A case study. In EASE 2012. 16th International Conference on Evaluation & Assessment in Software Engineering Proceedings. Ed. by Baldassarre, T., Genero, M., Mendes, E., and Piattini, M. DOI: 10.1049/ic.2012.0016 (cit. on pp. 40, 45, 48).

Kitchenham, B., Brereton, P., Li, Z., Budgen, D., and Burn, A. 2011. Repeatability of Systematic Literature Reviews. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 46–55. DOI: 10.1049/ic.2011.0006 (cit. on p. 48).

Kitchenham, B. and Charters, S. 2007. Guidelines for performing Systematic Literature Reviews in Software Engineering. Tech. rep. EBSE, July 2007. ⟨URL: http://www.dur.ac.uk/ebse/resources/guidelines/Systematic-reviews-5-8.pdf⟩ [visited on 2014-03-13] (cit. on pp. 40–43, 45–48, 68, 124).

Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O. P., Turner, M., Niazi, M., and Linkman, S. 2010. Systematic literature reviews in software engineering. A tertiary study. Information and Software Technology 52 (8), 792–805. DOI: 10.1016/j.infsof.2010.03.006 (cit. on p. 40).

Kleinschmager, S. 2012. Can static type systems speed up programming? An experimental evaluation of static and dynamic type systems. GRIN Verlag. ⟨URL: http://www.grin.com/en/e-book/199362/can-static-type-systems-speed-up-programming-an-experimental-evaluation⟩ (cit. on pp. 76, 79, 80, 178).

Kleinschmager, S., Hanenberg, S., Robbes, R., Tanter, E., and Stefik, A. 2012. Do static type systems improve the maintainability of software systems? An empirical study. In Program Comprehension (ICPC), 2012 IEEE 20th International Conference on, 153–162. DOI: 10.1109/ICPC.2012.6240483 (cit. on pp. 76, 79, 80, 178).

Kleinschmager, S. 2009. A Controlled Experiment for Measuring the Impact of Aspect-Oriented Programming on Software Development Time. GRIN Verlag. ⟨URL: http://www.grin.com/en/e-book/199337/a-controlled-experiment-for-measuring-the-impact-of-aspect-oriented-programming⟩ (cit. on pp. 76, 177).

Klerer, M. 1984. Experimental study of a two-dimensional language vs Fortran for first-course programmers. International Journal of Man-Machine Studies 20 (5), 445–467. DOI: 10.1016/S0020-7373(84)80021-8 (cit. on pp. 76, 178).

Klerer, M. 1991. Design of Very High-Level Computer Languages. A User-Oriented Approach. 2nd ed. 1991: McGraw-Hill (cit. on p. 37).

Knuth, D. E. and Trabb Pardo, L. 2003. The Early Development of Programming Languages. In Selected Papers on Computer Languages. Ed. by Knuth, D. E. CSLI Lecture Notes 139. Originally published in the Encyclopedia of Computer Science and Technology 7, 1977. Center for the Study of Language and Information, Stanford University, 1–93 (cit. on pp. 30, 34).

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., and Wiedenbeck, S. 2011. The State of the Art in End-user Software Engineering. ACM Computing Surveys 43 (3), a21. DOI: 10.1145/1922649.1922658 (cit. on p. 19).

Kosar, T., Oliveira, N., Mernik, M., Pereira, V. J. M., Črepinšek, M., Cruz, D. da, and Henriques, R. P. 2010. Comparing general-purpose and domain-specific languages: An empirical study. Computer Science and Information Systems 7 (2), 247–264. DOI: 10.2298/CSIS1002247K (cit. on pp. 76, 178).

Koss, A. M. 2003. Programming on the Univac 1. A Woman's Account. IEEE Annals of the History of Computing 25 (1), 48–59. DOI: 10.1109/MAHC.2003.1179879 (cit. on p. 20).

Koster, C. H. A. 1974. Two-level grammars. In Compiler Construction. An Advanced Course. Ed. by Bauer, F. L. and Eickel, J. Lecture Notes in Computer Science 21. Berlin: Springer, 146–156. DOI: 10.1007/978-3-662-21549-4_7 (cit. on p. 26).

Krishnamurthi, S. 2008. Teaching Programming Languages in a Post-Linnaean Age. ACM SIGPLAN Notices 43 (11). DOI: 10.1145/1480828.1480846 (cit. on p. 24).

Krogdahl, S. 2005. The birth of Simula. In History of Nordic Computing. IFIP WG9.7 First Working Conference on the History of Nordic Computing (HiNC1). Ed. by Bubenko Jr., J., Impagliazzo, J., and Sølvberg, A. IFIP International Federation for Information Processing 174. Springer, 261–275. DOI: 10.1007/0-387-24168-X_24. ⟨URL: http://home.ifi.uio.no/steinkr/papers/HiNC1-webversion-simula.pdf⟩ (cit. on p. 30).

Kuhn, T. S. 1996. The Structure of Scientific Revolutions. 3rd ed. University of Chicago Press (cit. on pp. 22, 92).

Kulesza, U., Sant'Anna, C., Garcia, A., Coelho, R., Staa, A. von, and Lucena, C. 2006. Quantifying the Effects of Aspect-Oriented Programming: A Maintenance Study. In Software Maintenance, 2006. ICSM '06. 22nd IEEE International Conference on, 223–233. DOI: 10.1109/ICSM.2006.48 (cit. on pp. 76, 178).

Kurtz, T. E. 1981. BASIC. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 515–537. DOI: 10.1145/800025.1198404 (cit. on p. 31).

Landin, P. J. 1965. A Correspondence Between ALGOL 60 and Church's Lambda Notation. Part I. Communications of the ACM 8 (2), 89–101. DOI: 10.1145/363744.363749 (cit. on p. 33).

Landis, J. R. and Koch, G. G. 1977. The Measurement of Observer Agreement for Categorical Data. Biometrics 33 (1), 159–174. ⟨URL: http://www.jstor.org/stable/2529310⟩ [visited on 2014-03-19] (cit. on p. 46).

Laughery Jr., K. R. and Laughery Sr., K. R. 1985. Human factors in software engineering: A review of the literature. Journal of Systems and Software 5 (1), 3–14. DOI: 10.1016/0164-1212(85)90003-2 (cit. on pp. 76, 180–182).

Launchbury, J. 1993. A Natural Semantics for Lazy Evaluation. In POPL'93. Proceedings of the 20th ACM SIGACT–SIGPLAN Symposium on Principles of Programming Languages, 144–154. DOI: 10.1145/158511.158618 (cit. on p. 17).

Leavenworth, B. M. and Sammet, J. E. 1974. An Overview of Nonprocedural Languages. In Proceedings of the ACM SIGPLAN Symposium on Very high level languages, 1–12. DOI: 10.1145/800233.807040 (cit. on p. 23).

Leblanc, R. J. and Fischer, C. N. 1982. A case study of run-time errors in Pascal programs. Software: Practice and Experience 12 (9), 825–834. DOI: 10.1002/spe.4380120903 (cit. on pp. 76, 85, 178).

Lee, K., LaMarca, A., and Chambers, C. 2003. HydroJ: object-oriented pattern matching for evolvable distributed systems. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications, 205–223. DOI: 10.1145/949305.949324 (cit. on pp. 76, 84, 85, 178).

Lewis, J. A., Henry, S. M., Kafura, D. G., and Schulman, R. S. 1991. An empirical study of the object-oriented paradigm and software reuse. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications, 184–196. DOI: 10.1145/117954.117969 (cit. on pp. 76, 178, 181).

Lewis, J. A., Henry, S. M., Kafura, D. G., and Schulman, R. S. 1992. On the Relationship Between the Object-Oriented Paradigm and Software Reuse: An Empirical Investigation. Technical Report TR-92-15. Computer Science, Virginia Polytechnic Institute and State University. ⟨URL: http://eprints.cs.vt.edu/archive/00000295/⟩ (cit. on pp. 76, 178, 181).

Levy, Y. and Ellis, T. J. 2006. A Systems Approach to Conduct an Effective Literature Review in Support of Information Systems Research. Informing Science Journal 9, 181–212. ⟨URL: http://www.inform.nu/Articles/Vol9/V9p181-212Levy99.pdf⟩ [visited on 2014-03-14] (cit. on p. 45).

Lima, A., Goulão, M., and Monteiro, M. P. 2011. Evidence-Based Comparison of Modularity Support Between Java and Object Teams. Paper at arXiv. ⟨URL: http://arxiv.org/abs/1109.2075⟩ (cit. on pp. 76, 178).

Lin, Y. and Blackburn, S. M. 2012. Bypassing Portability Pitfalls of High-level Low-level Programming. In VMIL'12. Proceedings of the 2012 ACM Workshop on Virtual Machines and Intermediate Languages, 23–32. DOI: 10.1145/2414740.2414746 (cit. on p. 20).

Lincoln, Y. S., Lynham, S. A., and Guba, E. G. 2011. Paradigmatic Controversies, Contradictions, and Emerging Confluences, Revisited. In The SAGE Handbook of Qualitative Research. Ed. by Denzin, N. K. and Lincoln, Y. S. 4th ed. Los Angeles: SAGE, 97–128 (cit. on p. 53).

Lind, J. 1757. A Treatise on the Scurvy in Three Parts. Containing An Inquiry into the Nature, Causes, and Cure, of that Disease together with a Critical and Chronological View of what has been published on the Subject. 2nd ed. London: A. Millar. ⟨URL: https://play.google.com/store/books/details?id=oP1UEXWU7fsC⟩ (cit. on p. 41).

Lipton, P. 2004. Inference to the Best Explanation. 2nd ed. London: Routledge (cit. on p. 53).

Liskov, B. and Zilles, S. 1974. Programming with Abstract Data Types. In Proceedings of the ACM SIGPLAN Symposium on Very high level languages, 50–59. DOI: 10.1145/800233.807045 (cit. on p. 31).

Liu, J., Kimball, A., and Myers, A. C. 2006. Interruptible iterators. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL), 283–294. DOI: 10.1145/1111037.1111063 (cit. on pp. 76, 85, 178).

Loui, R. P. 2008. In Praise of Scripting. Real Programming Pragmatism. Computer 41 (7), 22–26. DOI: 10.1109/MC.2008.228 (cit. on p. 24).

Lucas, H. C. and Kaplan, R. B. 1976. A Structured Programming Experiment. The Computer Journal 19 (2), 136–138. DOI: 10.1093/comjnl/19.2.136 (cit. on pp. 76, 79, 80, 178, 181).

Luff, M. 2009. Empirically investigating parallel programming paradigms: A null result. In Proc. PLATEAU 2009, 43–49. ⟨URL: http://ecs.victoria.ac.nz/foswiki/pub/Events/PLATEAU/2009Program/plateau09-luff.pdf⟩ (cit. on pp. 76, 81, 82, 178, 181).

MacDonell, S., Shepperd, M., Kitchenham, B., and Mendes, E. 2010. How Reliable Are Systematic Reviews in Empirical Software Engineering? IEEE Transactions on Software Engineering 36 (5), 676–687. DOI: 10.1109/TSE.2010.28 (cit. on p. 40).

Mackenzie, J. 2011. Positivism and Constructivism, Truth and 'Truth'. Educational Philosophy and Theory 43 (5), 534–546. DOI: 10.1111/j.1469-5812.2010.00676.x (cit. on p. 52).

Madeyski, L. and Szala, L. 2007. Impact of aspect-oriented programming on software development efficiency and design quality: an empirical study. Software, IET 1 (5), 180–187. DOI: 10.1049/iet-sen:20060071 (cit. on pp. 76, 178).

Mahoney, M. S. 2008. What Makes the History of Software Hard. IEEE Annals of the History of Computing 30 (3), 8–18. DOI: 10.1109/MAHC.2008.55 (cit. on p. 34).

Malayeri, D. and Aldrich, J. 2009. Is Structural Subtyping Useful? An Empirical Study. In Programming Languages and Systems. Ed. by Castagna, G. Vol. 5502. Lecture Notes in Computer Science, 95–111. DOI: 10.1007/978-3-642-00590-9_8 (cit. on pp. 62, 76, 84, 85, 89, 178).

Malheiros, V., Höhn, E., Pinho, R., Mendonça, M., and Maldonado, J. C. 2007. A Visual Text Mining approach for Systematic Reviews. In ESEM 2007. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, 245–254. DOI: 10.1109/ESEM.2007.21 (cit. on p. 47).

Marks, S. L. 1982. JOSS. Conversational Computing for the Nonprogrammer. Annals of the History of Computing 4 (1), 35–52. DOI: 10.1109/MAHC.1982.10004 (cit. on p. 34).

Markstrum, S. 2010. Staking Claims. A History of Programming Language Design Claims and Evidence. In PLATEAU '10. Evaluation and Usability of Programming Languages and Tools. DOI: 10.1145/1937117.1937124 (cit. on pp. 13, 37).

Marlow, S., ed. Haskell 2010 Language Report 2010. ⟨URL: http://www.haskell.org/onlinereport/haskell2010⟩ (cit. on p. 18).

Marshall, C. and Brereton, P. 2013. Tools to Support Systematic Literature Reviews in Software Engineering. A Mapping Study. In ESEM 2013. 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement Proceedings, 296–299. DOI: 10.1109/ESEM.2013.32 (cit. on p. 42).

Martin, J. 1982. Application Development Without Programmers. Englewood Cliffs: Prentice-Hall (cit. on p. 21).

Martin, J. 1985. Fourth-Generation Languages. Vol. I: Principles. Englewood Cliffs: Prentice–Hall (cit. on p. 21).

Mayer, C., Hanenberg, S., Robbes, R., Tanter, É., and Stefik, A. 2012a. Static type systems (sometimes) have a positive impact on the usability of undocumented software: An empirical evaluation. Technical report. ⟨URL: http://www.dcc.uchile.cl/TR/2012/TR_DCC-20120418-005.pdf⟩ (cit. on pp. 76, 79, 80, 178).

Mayer, C., Hanenberg, S., Robbes, R., Tanter, É., and Stefik, A. 2012b. An empirical study of the influence of static type systems on the usability of undocumented software. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM, 683–702. DOI: 10.1145/2384616.2384666 (cit. on pp. 37, 76, 79, 80, 178).

McCaffrey, J. D. and Bonar, A. 2010. A Case Study of the Factors Associated with Using the F# Programming Language for Software Test Automation. In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on, 1009–1013. DOI: 10.1109/ITNG.2010.253 (cit. on pp. 76, 178).

McCarthy, J. 1996. Towards a Mathematical Science of Computation. ⟨URL: http://www-formal.stanford.edu/jmc/towards.html⟩ [visited on 2014-04-11] (cit. on p. 25).

McCarthy, J. 1959. Letter to the editor. Communications of the ACM 2 (8), 2–3. DOI: perlis58:_prelim_repor (cit. on p. 27).

McCarthy, J. 1960. Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I. Communications of the ACM 3 (4), 184–195. DOI: 10.1145/367177.367199 (cit. on pp. 31, 33).

McCarthy, J. 1964. Definition of New Data Types in ALGOL X. ALGOL Bulletin (18) (Oct. 1964), 45. ⟨URL: http://dl.acm.org/citation.cfm?id=1060993⟩ (cit. on p. 30).

McEwan, P., Bergenheim, K., Yuan, Y., Tetlow, A. P., and Gordon, J. P. 2010. Assessing the Relationship between Computational Speed and Precision: A Case Study Comparing an Interpreted versus Compiled Programming Language using a Stochastic Simulation Model in Diabetes Care. PharmacoEconomics 28 (8), 665–674. ⟨URL: http://www.ingentaconnect.com/content/adis/pec/2010/00000028/00000008/art00005⟩ (cit. on pp. 76, 178).

McIver, L. 2000. The Effect of Programming Language on Error Rates of Novice Programmers. In PPIG 2000. ⟨URL: http://www.ppig.org/papers/12th-mciver.pdf⟩ (cit. on pp. 76, 178).

Meek, B. 1990. The static semantics file. SIGPLAN Notices 25 (4), 33–42. DOI: 10.1145/987481.987483 (cit. on p. 26).

Meyerovich, L. A. and Rabkin, A. 2012. Socio-PLT. Principles for Programming Language Adoption. In Onward! 2012. Proceedings of the ACM Intenational Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, 39–53. DOI: 10.1145/2384592.2384597 (cit. on p. 36).

Miara, R. J., Musselman, J. A., Navarro, J. A., and Shneiderman, B. 1983. Program indentation and comprehensibility. Communications of the ACM 26 (11), 861–867. DOI: 10.1145/182.358437 (cit. on pp. 76, 81, 82, 178).

Mikkonen, T. and Taivalsaari, A. 2008. Using JavaScript as a Real Programming Language. Tech. rep. SMLI TR-2007-168. Sun Microsystems Laboratories, Oct. 2008. ⟨URL: http://dl.acm.org/citation.cfm?id=1698202⟩ (cit. on p. 31).

Miller, A., Hicks, M., Katz, J., and Shi, E. 2014. Authenticated Data Structures, Generically. In POPL'14. Proceedings of the 41st Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming languages, 411–423. DOI: 10.1145/2535838.2535851 (cit. on p. 13).

Millstein, T. 2004. Practical predicate dispatch. In OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 345–364. DOI: 10.1145/1028976.1029006 (cit. on pp. 76, 85, 179).

Millstein, T., Frost, C., Ryder, J., and Warth, A. 2009. Expressive and modular predicate dispatch for Java. ACM Transactions on Programming Languages and Systems 31 (2). DOI: 10.1145/1462166.1462168 (cit. on pp. 76, 85, 179).

Milner, R. 1978. A Theory of Type Polymorphism in Programming. Journal of Computer and System Sciences 17 (3), 348–375. DOI: 10.1016/0022-0000(78)90014-4 (cit. on p. 33).

Morris Jr., J. H. 1969. Lambda-calculus Models of Programming Languages. PhD thesis. Massachusetts Institute of Technology. ⟨URL: http://hdl.handle.net/1721.1/64850⟩ [visited on 2014-04-21] (cit. on p. 33).

Morrison, A., Viller, S., and Mitchell, P. 2010. Ethnography Considered Useful. Situating criticality. In OZCHI 2010. Conference Proceedings, 184–187. DOI: 10.1145/1952222.1952261 (cit. on p. 49).

Mortensen, M., Ghosh, S., and Bieman, J. M. 2012. Aspect-Oriented Refactoring of Legacy Applications: An Evaluation. Software Engineering, IEEE Transactions on 38 (1), 118–140. DOI: 10.1109/TSE.2010.109 (cit. on pp. 76, 179).

Mosses, P. D. 2000. A Foreword to 'Fundamental Concepts in Programming Languages'. Higher-Order and Symbolic Computation 13 (1–2), 7–9. DOI: 10.1023/A:1010048229036 (cit. on p. 30).

Mosses, P. D. 2001. The Varieties of Programming Language Semantics. And Their Uses. In Perspectives of System Informatics. 4th International Andrei Ershov Memorial Conference, PSI 2001. Lecture Notes in Computer Science 2244. Berlin: Springer, 165–190. DOI: 10.1007/3-540-45575-2_18 (cit. on p. 26).

Myers, B. A., Giuse, D. A., and Zanden, B. V. 1992. Declarative programming in a prototype-instance system: object-oriented programming without writing methods. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications, 184–200. DOI: 10.1145/141936.141953 (cit. on pp. 76, 179).

Myers, B. A., Ko, A. J., Park, S. Y., Stylos, J., LaToza, T. D., and Beaton, J. 2008. More natural end-user software engineering. In Proceedings of the 4th international workshop on End-user software engineering. WEUSE '08. Leipzig, Germany: ACM, 30–34. ISBN: 978-1-60558-034-0. DOI: 10.1145/1370847.1370854 (cit. on p. 37).

Myers, B. A., Pane, J. F., and Ko, A. 2004. Natural programming languages and environments. Communications of the ACM 47 (9) (Sept. 2004), 47–52. ISSN: 0001-0782. DOI: 10.1145/1015864.1015888 (cit. on pp. 13, 15, 37).

Myrtveit, I. and Stensrud, E. 2008. An empirical study of software development productivity in C and C++. In Proc. Norsk Informatikkonferanse 2008. ⟨URL: http://www.nik.no/2008/03-Myrtveit.pdf⟩ (cit. on pp. 76, 179).

Nanz, S., Torshizi, F., Pedroni, M., and Meyer, B. 2011a. Design of an Empirical Study for Comparing the Usability of Concurrent Programming Languages. In Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on, 325–334. DOI: 10.1109/ESEM.2011.41 (cit. on pp. 76, 79, 80, 179).

Nanz, S., Torshizi, F., Pedroni, M., and Meyer, B. 2011b. Empirical assessment of languages for teaching concurrency: Methodology and application. In Software Engineering Education and Training (CSEE T), 2011 24th IEEE-CS Conference on, 477–481. DOI: 10.1109/CSEET.2011.5876128 (cit. on pp. 76, 79, 80, 179).

Nanz, S., Torshizi, F., Pedroni, M., and Meyer, B. 2010. A Comparative Study of the Usability of Two Object-oriented Concurrent Programming Languages. Paper in arXiv. ⟨URL: http://arxiv.org/abs/1011.6047⟩ (cit. on pp. 76, 79, 80, 179).

Naur, P. 1981. The European side of the last phase development of ALGOL 60. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 92–139. DOI: 10.1145/800025.1198353 (cit. on p. 35).

Naur, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samuelson, K., Vauquois, B., Wegstein, J. H., Wijngaarden, A. van, and Woodger, M. 1960. Report on the Algorithmic Language ALGOL 60. Communications of the ACM 3 (5), 299–314. DOI: 10.1145/367236.367262 (cit. on pp. 25, 27, 29).

Naur, P., Backus, J. W., Bauer, F. L., Green, J., Katz, C., McCarthy, J., Perlis, A. J., Rutishauser, H., Samuelson, K., Vauquois, B., Wegstein, J. H., Wijngaarden, A. van, and Woodger, M. 1963. Revised Report on the Algorithmic Language ALGOL 60. Communications of the ACM 6 (1), 1–17. DOI: 10.1145/366193.366201 (cit. on pp. 27, 29).

Necula, G. C., Condit, J., Harren, M., McPeak, S., and Weimer, W. 2005. CCured: type-safe retrofitting of legacy software. ACM Transactions on Programming Languages and Systems 27 (3), 477–526. DOI: 10.1145/1065887.1065892 (cit. on pp. 76, 179).

Nichols, H. 1891. The Psychology of Time. American Journal of Psychology 3 (4), 453–529. DOI: 10.2307/1412061 (cit. on p. 41).

Nieminen, P., Pölönen, I., and Sipola, T. 2013. Research literature clustering using diffusion maps. Journal of Informetrics 7 (4), 874–886. DOI: 10.1016/j.joi.2013.08.004 (cit. on p. 48).

Nofre, D. 2010. Unraveling Algol. US, Europe, and the Creation of a Programming Language. IEEE Annals of the History of Computing 32 (2), 58–68. DOI: 10.1109/MAHC.2010.4 (cit. on pp. 34, 35).

Norcio, A. F. 1982. Indentation, documentation and programmer comprehension. In Proceedings of the 1982 Conference on Human Factors in Computing Systems. CHI '82. New York, NY, USA: ACM, 118–120. DOI: 10.1145/800049.801766 (cit. on pp. 76, 79, 80, 179, 181).

Nystrom, N., Qi, X., and Myers, A. C. 2006. J&: nested intersection for scalable software composition. In OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications, 21–36. DOI: 10.1145/1167473.1167476 (cit. on pp. 76, 85, 179).

Nyström, J., Trinder, P., and King, D. 2007. Evaluating high-level distributed language constructs. In Proceedings of the 12th ACM SIGPLAN international conference on Functional programming. ICFP '07. New York, NY, USA: ACM, 203–212. DOI: 10.1145/1291151.1291182 (cit. on pp. 76, 179).

O'Regan, G. 2012. A Brief History of Computing. 2nd ed. London: Springer. DOI: 10.1007/978-1-4471-2359-0 (cit. on p. 21).

O'Rourke, K. 2007. An historical perspective on meta-analysis. Dealing quantitatively with varying study results. Journal of the Royal Society of Medicine 100 (12). DOI: 10.1258/jrsm.100.12.579 (cit. on p. 40).

Ousterhout, J. K. 1998. Scripting. Higher-Level Programming for the 21st Century. Computer 31 (3), 23–30. DOI: 10.1109/2.660187 (cit. on p. 23).

Pair, C. 1990. Programming, Programming Languages and Programming Methods. In Psychology of Programming. Ed. by Hoc, J.-M., Green, T. R. G., Samurçay, R., and Gilmore, D. J. London: Academic, 9–19 (cit. on pp. 18, 19).

Pane, J. F., Myers, B. A., and Miller, L. B. 2002. Using HCI techniques to design a more usable programming system. In Proc. Human Centric Computing Languages and Environments, 198–206. DOI: 10.1109/HCC.2002.1046372 (cit. on p. 37).

Pane, J. F. and Myers, B. A. 2000. The influence of the psychology of programming on a language design: Project status report. In PPIG 2000. ⟨URL: http://ppig.org/papers/12th-pane.pdf⟩ (cit. on pp. 37, 76, 180, 182).

Pane, J. F. and Myers, B. A. 2006. More Natural Programming Languages and Environments. In End User Development. Ed. by Lieberman, H., Paternò, F., and Wulf, V. Vol. 9. Human–Computer Interaction Series. Springer Netherlands, 31–50. DOI: 10.1007/1-4020-5386-X_3 (cit. on pp. 37, 76, 180, 182).

Pane, J. and Myers, B. 1996. Usability Issues in the Design of Novice Programming Systems. Paper 820. Institute for Software Research. ⟨URL: http://repository.cmu.edu/isr/820⟩ (cit. on p. 37).

Pankratius, V., Schmidt, F., and Garreton, G. 2012. Combining functional and imperative programming for multicore software: An empirical study evaluating Scala and Java. In Software Engineering (ICSE), 2012 34th International Conference on, 123–133. DOI: 10.1109/ICSE.2012.6227200 (cit. on pp. 70, 76, 179).

Pankratius, V. and Adl-Tabatabai, A.-R. 2011. A study of transactional memory vs. locks in practice. In Proceedings of the 23rd ACM symposium on Parallelism in algorithms and architectures. SPAA '11. New York, NY, USA: ACM, 43–52. DOI: 10.1145/1989493.1989500 (cit. on pp. 76, 79, 80, 179).

Pankratius, V., Adl-Tabatabai, A.-R., and Otto, F. 2009. Does Transactional Memory Keep Its Promises? Results from an Empirical Study. Technical Report 2009-12. IPD, University of Karlsruhe, Germany. ⟨URL: http://www.rz.uni-karlsruhe.de/~kb95/papers/pankratius-TMStudy.pdf⟩ (cit. on pp. 76, 79, 80, 179).

paradigm, n. 2014. In OED Online. Mar. 2014. ⟨URL: http://www.oed.com/view/Entry/137329⟩ [visited on 2014-04-10] (cit. on p. 22).

Patel, I. and Gilbert, J. R. 2008. An empirical study of the performance and productivity of two parallel programming models. In Parallel and Distributed Processing, 2008. IPDPS 2008. IEEE International Symposium on, 1–7. DOI: 10.1109/IPDPS.2008.4536192 (cit. on pp. 76, 179).

Paterson, L. L. 2011. Epicene pronouns in UK national newspapers. A diachronic study. ICAME Journal 35, 171–184. ⟨URL: http://icame.uib.no/ij35/Laura_Louise_Paterson.pdf⟩ (cit. on p. 6).

Patterson, D. A. 1981. An experiment in high level language microprogramming and verification. Communications of the ACM 24 (10), 699–709. DOI: 10.1145/358769.358788 (cit. on pp. 76, 179).

Penzenstandler, B., Bauer, V., Calero, C., and Franch, X. 2012. Sustainability in Software Engineering. A Systematic Literature Review. In Proc. 16th International Conference on Evaluation & Assessment in Software Engineering (EASE), 32–41. DOI: 10.1049/ic.2012.0004 (cit. on p. 40).

Perlis, A. J. and Samelson, K. 1958. Preliminary Report. International Algebraic Language. Communications of the ACM 1 (12), 8–22. DOI: 10.1145/377924.594925 (cit. on pp. 27, 29).

Perlis, A. J. 1981. The American side of the development of ALGOL. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 75–91. DOI: 10.1145/800025.1198352 (cit. on p. 35).

perlsyn. Perl syntax 2014. Version 5.18.2. ⟨URL: http://perldoc.perl.org/perlsyn.pdf⟩ [visited on 2014-04-16] (cit. on pp. 28, 29).

Perrott, R. H., Raja, A. K., and O'Kane, P. C. 1980. A simulation experiment using two languages. The Computer Journal 23 (2), 142–146. DOI: 10.1093/comjnl/23.2.142 (cit. on pp. 76, 179).

Petersen, K. and Ali, N. B. 2011. Identifying Strategies for Study Selection in Systematic Reviews and Maps. In Proceedings of the 2011 Fifth International Symposium on Empirical Software Engineering and Measurement. DOI: 10.1109/ESEM.2011.46 (cit. on p. 46).

Petersen, K., Feldt, R., Mujtaba, S., and Mattson, M. 2008. Systematic Mapping Studies in Software Engineering. In Proc. 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). ⟨URL: http://ewic.bcs.org/content/ConWebDoc/19543⟩ [visited on 2014-03-13] (cit. on pp. 41–43, 45, 47).

Petticrew, M. and Roberts, H. 2006. Systematic Reviews in the Social Sciences. A Practical Guide. Malden, MA: Blackwell (cit. on pp. 41–46).

Pierce, B. C. 2002. Types and Programming Languages. Cambridge, Massachusetts: MIT Press (cit. on pp. 25, 31–33).

Pigott, D. 2006. HOPL. An interactive Roster of Programming Languages. ⟨URL: https://web.archive.org/web/20111205165034/http://hopl.murdoch.edu.au/⟩ (cit. on p. 13).

Pinsky, M. L. and Palumbi, S. R. 2014. Meta-analysis reveals lower genetic diversity in overfished populations. Molecular Ecology 23 (1), 29–39. DOI: 10.1111/mec.12509 (cit. on p. 40).

Pogran, K., Vittal, J., Crocker, D., and Henderson, A. 1977. Proposed Official Standard for the Format of ARPA Network Messages. Request for Comments 724. ⟨URL: http://www.ietf.org/rfc/rfc724.txt⟩ (cit. on p. 149).

Poletto, M., Hsieh, W. C., Engler, D. R., and Kaashoek, M. F. 1999. C and tcc: a language and compiler for dynamic code generation. ACM Transactions on Programming Languages and Systems 21 (2), 324–369. DOI: 10.1145/316686.316697 (cit. on pp. 76, 179).

POPL'14. Proceedings of the 41st Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming languages 2014.

Popper, K. R. 1980. The Logic of Scientific Discovery. 10th impression (revised). London: Unwin Hyman (cit. on p. 52).

Prechelt, L. 2000. An empirical comparison of seven programming languages. Computer 33 (10), 23–29. DOI: 10.1109/2.876288 (cit. on pp. 76, 89, 179).

Prechelt, L. and Tichy, W. F. 1996. An experiment to assess the benefits of intermodule type checking. In Software Metrics Symposium, 1996., Proceedings of the 3rd International, 112–119. DOI: 10.1109/METRIC.1996.492448 (cit. on pp. 76, 79, 80, 89, 179).

Prechelt, L. and Tichy, W. F. 1998. A controlled experiment to assess the benefits of procedure argument type checking. Software Engineering, IEEE Transactions on 24 (4), 302–312. DOI: 10.1109/32.677186 (cit. on pp. 62, 76, 79, 80, 89, 179).

Prechelt, L. 2003. Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java. Advances in Computers 57, 205–270. DOI: 10.1016/S0065-2458(03)57005-X (cit. on pp. 76, 89, 179).

Prechelt, L., Unger, B., Philippsen, M., and Tichy, W. 2003. A controlled experiment on inheritance depth as a cost factor for code maintenance. Journal of Systems and Software 65 (2), 115–126. DOI: 10.1016/S0164-1212(02)00053-5 (cit. on pp. 76, 79, 80, 179).

Priestley, M. 2011. A Science of Operations. Machines, Logic and the Invention of Programming. London: Springer. DOI: 10.1007/978-1-84882-555-0 (cit. on p. 22).

Programming for the UNIVAC Fac-Tronic System 1953. Remington–Rand Eckert–Mauchly Division. Jan. 1953. ⟨URL: http://www.bitsavers.org/pdf/univac/univac1/UNIVAC_Programming_Jan53.pdf⟩ [visited on 2014-04-13] (cit. on p. 20).

programming, n. 2013. In OED Online. Dec. 2013. ⟨URL: http://www.oed.com/view/Entry/152232⟩ [visited on 2014-02-03] (cit. on p. 16).

Przybyłek, A. 2011. Where the Truth Lies: AOP and Its Impact on Software Modularity. In Fundamental Approaches to Software Engineering. Ed. by Giannakopoulou, D. and Orejas, F. Vol. 6603. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 447–461. DOI: 10.1007/978-3-642-19811-3_31 (cit. on pp. 76, 179).

Pullum, G. K. 2014. Fear and loathing of the English passive. Language & Communication. In press. DOI: 10.1016/j.langcom.2013.08.009 (cit. on p. 6).

Qi, X. and Myers, A. C. 2010. Homogeneous family sharing. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications, 520–538. DOI: 10.1145/1869459.1869502 (cit. on pp. 76, 85, 179).

Quine, W. V. 1951. Two dogmas of empiricism. Philosophical Review 60 (1), 20–43. ⟨URL: http://www.jstor.org/stable/2181906⟩ (cit. on p. 52).

Radin, G. 1981. The Early History and Characteristics of PL/I. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 551–575. DOI: 10.1145/800025.1198410 (cit. on p. 30).

Ramalingam, V. and Wiedenbeck, S. 1997. An empirical study of novice program comprehension in the imperative and object-oriented styles. In Papers presented at the seventh workshop on Empirical studies of programmers, 124–139. DOI: 10.1145/266399.266411 (cit. on pp. 76, 179, 181).

Ramesh, V., Glass, R. L., and Vessey, I. 2004. Research in computer science. An empirical study. Journal of Systems and Software 70, 165–176 (cit. on p. 59).

Ramsey, F. P. 1926. The Foundations of Mathematics. Proceedings of the London Mathematical Society. 2nd ser. 25 (1), 338–384. DOI: 10.1112/plms/s2-25.1.338 (cit. on p. 32).

Rawlings, N. 2014. The History of NOMAD. A Fourth Generation Language. IEEE Annals of the History of Computing 36 (1), 30–38. DOI: 10.1109/MAHC.2014.10 (cit. on p. 21).

Resnick, P., ed. Internet Message Format 2008. Request for Comments 5322. ⟨URL: http://www.ietf.org/rfc/rfc5322.txt⟩ (cit. on p. 149).

Reynolds, J. C. 1974. Towards a Theory of Type Structure. In Programming Symposium. Proceedings, Colloque sur la Programmation. Ed. by Robinet, B. Lecture Notes in Computer Science 19. Berlin: Springer, 408–425. DOI: 10.1007/3-540-06859-7_148 (cit. on p. 33).

Reynolds, J. C. 1985. Three Approaches to Type Structure. In Mathematical Foundations of Software Development. Proceedings of the International Joint Conference on Theory and Practice of Software Development (TAPSOFT) Berlin, March 25–29, 1985 Volume I: Colloquium on Trees in Algebra and Programming (CAAP'85). Ed. by Ehrig, H., Floyd, C., Nivat, M., and Thatcher, J. Lecture Notes in Computer Science 185. Berlin: Springer, 97–138. DOI: 10.1007/3-540-15198-2_7 (cit. on p. 33).

Reynolds, J. C. 1998. Theories of Programming Languages. Cambridge University Press (cit. on p. 17).

Ritchie, D. M. 1974. C Reference Manual. Bell Laboratories. ⟨URL: http://cm.bell-labs.com/cm/cs/who/dmr/cman74.pdf⟩ [visited on 2014-04-17] (cit. on pp. 29, 157).

Roberts, E. S. 1995. Loop exits and structured programming: reopening the debate. In Proceedings of the twenty-sixth SIGCSE technical symposium on Computer science education. SIGCSE '95. New York, NY, USA: ACM, 268–272. DOI: 10.1145/199688.199815 (cit. on pp. 76, 180–182).

Rosen, S. 1964. Programming Systems and Languages. A Historical Survey. In 1964 Spring Joint Computer Conference. AFIPS Conference Proceedings 25. Spartan, 1–15. DOI: 10.1145/1464122.1464124 (cit. on p. 34).

Rosen, S. 1972. Programming Systems and Languages 1965–1975. Communications of the ACM 15 (7), 591–600. DOI: 10.1145/361454.361482 (cit. on p. 34).

Rosenberg, W. and Donald, A. 1995. Evidence based medicine. An approach to clinical problem-solving. BMJ 310, 1122–1126. ⟨URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC2549505/⟩ [visited on 2011-09-26] (cit. on p. 38).

Ross, D. T. and Rodriguez, J. E. 1963. Theoretical Foundations for the Computer-Adided Design System. In 1963 Spring Joint Computer Conference. AFIPS Conference Proceedings 23. Spartan. DOI: 10.1145/1461551.1461589 (cit. on p. 30).

Rossbach, C. J., Hofmann, O. S., and Witchel, E. 2009. Is transactional programming actually easier? In Proc. 8th Annual Workshop on Duplicating, Deconstructing, and Debunking. ⟨URL: http://pharm.ece.wisc.edu/wddd/2009/papers/wddd_04.pdf⟩ (cit. on pp. 76, 79, 80, 179).

Rossbach, C. J., Hofmann, O. S., and Witchel, E. 2010. Is transactional programming actually easier? In Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. PPoPP '10. New York, NY, USA: ACM, 47–56. DOI: 10.1145/1693453.1693462 (cit. on pp. 76, 79, 80, 179, 181).

Rousseau, D. M. 2006. 2005 Presidential Address. Is There Such a Thing as "Evidence-Based Management"? Academy of Management Review 31 (2), 256–269. ⟨URL: http://search.ebscohost.com/login.aspx?direct=true&db=bsh&AN=20208679&site=ehost-live⟩ [visited on 2011-10-04] (cit. on p. 38).

Runeson, P., Höst, M., Rainer, A., and Regnell, B. 2012. Case Study Research in Software Engineering. Guidelines and Examples. Hoboken, New Jersey: Wiley (cit. on pp. 49, 90).

Russell, B. 1908. Mathematical Logic as based on the Theory of Types. American Journal of Mathematics 30 (3), 222–262. DOI: 10.2307/2369948 (cit. on p. 32).

Russell, B. 1983. The Problems of Philosophy. 11th impression. Oxford University Press (cit. on p. 52).

Russell, B. s.d. The Principles of Mathematics. 2nd ed. New York: Norton. ⟨URL: https://archive.org/details/principlesofmath005807mbp⟩ [visited on 2014-04-19] (cit. on p. 32).

Ryder, B. G., Soffa, M. L., and Burnett, M. 2005. The impact of software engineering research on modern progamming languages. ACM Transactions on Software Engineering and Methodology 14 (4) (Oct. 2005), 431–477. DOI: 10.1145/1101815.1101818 (cit. on pp. 34, 37).

Saal, H. J. and Weiss, Z. 1977. An empirical study of APL programs. Computer Languages 2 (3), 47–59. DOI: 10.1016/0096-0551(77)90007-8 (cit. on pp. 76, 179).

Sackman, H., Erikson, W. J., and Grant, E. E. 1968. Exploratory Experimental Studies Comparing Online and Offline Programming Performance. Communications of the ACM 11 (1), 3–11. DOI: 10.1145/362851.362858 (cit. on p. 36).

Sackman, H. 1970. Man–Computer Problem Solving. Experimental Evaluation of Time-Sharing and Batch Processing. Princeton: Auerbach (cit. on p. 36).

Sadowski, C. and Shewmaker, A. 2010. The last mile: parallel programming and usability. In Proceedings of the FSE/SDP workshop on Future of software engineering research. FoSER '10. New York, NY, USA: ACM, 309–314. DOI: 10.1145/1882362.1882426 (cit. on pp. 76, 180–182).

Sakkinen, M. 1992. Inheritance and Other Main Principles of C++ and Other Object-oriented Languages. Jyväskylä Studies in Computer Science, Economics and Statistics 20. University of Jyväskylä. ⟨URL: http://urn.fi/URN:ISBN:978-951-39-4352-3⟩ (cit. on p. 26).

Sammet, J. E. 1969. Programming Languages. History and Fundamentals. Englewood Cliffs: Prentice–Hall (cit. on pp. 17, 20, 23, 30, 34).

Sammet, J. E. 1972. Programming Languages. History and Future. Communications of the ACM 15 (7), 601–610. DOI: 10.1145/361454.361485 (cit. on p. 34).

Sammet, J. E. 1981. The early history of COBOL. In History of Programming Languages. Ed. by Wexelblat, R. L. ACM monograph series. New York, NY: Academic, 199–243. DOI: 10.1145/800025.1198367 (cit. on pp. 30, 35).

Sammet, J. E. 1991. Some Approaches to, and Illustrations of, Programming Language History. Annals of the History of Computing 13 (3), 33–50. DOI: 10.1109/MAHC.1991.10001 (cit. on p. 34).

Santos, R. E. S. and Silva, F. Q. B. da 2013. Motivation to Perform Systematic Reviews and their Impact on Software Engineering Practice. In Proc. 2013 ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), 292–295. DOI: 10.1109/ESEM.2013.36 (cit. on p. 40).

Sawadpong, P., Allen, E. B., and Williams, B. J. 2012. Exception Handling Defects: An Empirical Study. In High-Assurance Systems Engineering (HASE), 2012 IEEE 14th International Symposium on, 90–97. DOI: 10.1109/HASE.2012.24 (cit. on pp. 76, 85, 179).

Scholte, T., Robertson, W., Balzarotti, D., and Kirda, E. 2012. An empirical analysis of input validation mechanisms in web applications and languages. In Proceedings of the 27th Annual ACM Symposium on Applied Computing. New York, NY, USA: ACM, 1419–1426. DOI: 10.1145/2245276.2232004 (cit. on pp. 76, 85, 179).

Schwandt, T. A. 2007. The SAGE Dictionary of Qualitative Inquiry. SAGE. DOI: 10.4135/9781412986281 (cit. on p. 48).

Seixas, N., Fonseca, J., Vieira, M., and Madeira, H. 2009. Looking at Web Security Vulnerabilities from the Programming Language Perspective: A Field Study. In Software Reliability Engineering, 2009. ISSRE '09. 20th International Symposium on, 129–135. DOI: 10.1109/ISSRE.2009.30 (cit. on pp. 70, 76, 84, 85, 88, 179).

Sethi, R. 1996. Programming Languages. Concepts & Constructs. 2nd ed. Reading, MA: Addison–Wesley (cit. on p. 16).

Shaikh, W., Vayda, E., and Feldman, W. 1976. A Systematic Review of the Literature on Evaluative Studies of Tonsillectomy and Adenoidectomy. Pediatrics 57 (3), 401–407. ⟨URL: http://pediatrics.aappublications.org/content/57/3/401⟩ [visited on 2014-05-05] (cit. on p. 41).

Shapiro, S. 1997. Splitting the Difference. The Historical Necessity of Synthesis in Software Engineering. IEEE Annals of the History of Computing 19 (1), 20–54. DOI: 10.1109/85.560729 (cit. on p. 34).

Sheil, B. A. 1981. The Psychological Study of Programming. ACM Computing Surveys 13 (1), 101–120. DOI: 10.1145/356835.356840 (cit. on pp. 31, 36, 37, 76, 92, 98, 180–182).

Sheppard, S. B., Curtis, B., Milliman, P., Borst, M. A., and Love, T. 1979. First-year results from a research program on human factors in software engineering. Managing Requirements Knowledge, International Workshop on, 1021. DOI: 10.1109/AFIPS.1979.59 (cit. on pp. 76, 79, 80, 179, 181).

Sherman, L. W. 1998. Evidence-Based Policing. Ideas in American Policing. Police Foundation. ⟨URL: http://www.policefoundation.org/pdf/Sherman.pdf⟩ [visited on 2011-10-04] (cit. on p. 38).

Shneiderman, B. 1975. Experimental testing in programming languages, stylistic considerations and design techniques. In Proceedings of the May 19-22, 1975, national computer conference and exposition, 653–656. DOI: 10.1145/1499949.1500087 (cit. on pp. 76, 180, 181).

Shneiderman, B. 1976. Exploratory experiments in programmer behavior. International Journal of Computer and Information Sciences 5 (2), 123–143. DOI: 10.1007/BF00975629 (cit. on pp. 76, 81, 82, 179, 181).

Shneiderman, B. 1980. Software Psychology. Human Factors in Computer and Information Systems. Cambridge, MA: Winthrop (cit. on pp. 13, 36).

Shneiderman, B. 1985. The Relationship Between COBOL and Computer Science. Annals of the History of Computing 7 (4), 348–352. DOI: 10.1109/MAHC.1985.10041 (cit. on p. 30).

Shneiderman, B. and Mayer, R. 1979. Syntactic/semantic interactions in programmer behavior: A model and experimental results. International Journal of Parallel Programming 8 (3), 219–238. DOI: 10.1007/BF00977789 (cit. on pp. 76, 81, 82, 179).

Silva, F. Q. B. da, Santos, A. L. M., Soares, S. C. B., França, A. C. C., and Monteiro, C. V. F. 2010. A Critical Appraisal of Systematic Reviews in Software Engineering from the Perspective of the Research Questions Asked in the Reviews. In ESEM 2010. Proceedings of the 2010 ACM–IEEE International Symposium on Proceedings of the 2010 ACM-IEEE International Symposium on. DOI: 10.1145/1852786.1852830 (cit. on p. 40).

Silva, F. Q. B. da, Santos, A. L. M., Soares, S., França, A. C. C., Monteiro, C. V. F., and Maciel, F. F. 2011. Six years of systematic literature reviews in software engineering. An updated tertiary study. Information and Software Technology 53 (9), 899–913. DOI: 10.1016/j.infsof.2011.04.004 (cit. on p. 40).

Sime, M. E., Arblaster, A. T., and Green, T. R. G. 1977. Structuring the programmer's task. Journal of Occupational Psychology 50 (3), 205–216. DOI: 10.1111/j.2044-8325.1977.tb00376.x (cit. on pp. 76, 180–182).

Sime, M. E., Green, T. R. G., and Guest, D. J. 1973. Psychological evaluation of two conditional constructions used in computer languages. International Journal of Man-Machine Studies 5 (1), 105–113. DOI: 10.1016/S0020-7373(73)80011-2. ⟨URL: http://www.sciencedirect.com/science/article/pii/S0020737373800112⟩ (cit. on pp. 76, 81, 82, 88, 92, 179, 181).

Sime, M. E., Green, T. R. G., and Guest, D. J. 1977. Scope marking in computer conditionals – a psychological evaluation. International Journal of Man-Machine Studies 9 (1), 107–118. DOI: 10.1016/S0020-7373(77)80045-X (cit. on pp. 28, 76, 81, 82, 88, 179, 181).

Sime, M. E., Green, T. R. G., and Guest, D. J. 1999. Psychological Evaluation of Two Conditional Constructions Used in Computer Languages. International Journal of Human-Computer Studies 51 (2), 125–133. DOI: 10.1006/ijhc.1972.0302 (cit. on pp. 28, 76, 81, 82, 88, 179).

Simmonds, D. M. 2012. The Programming Paradigm Evolution. Computer 45 (6), 93–95 (cit. on p. 23).

Skoglund, M. and Runeson, P. 2009. Reference-Based Search Strategies in Systematic Reviews. In Proceedings of the 13th International Conference on Evaluation and Assessment in Software Engineering (EASE). ⟨URL: http://ewic.bcs.org/content/ConWebDoc/25022⟩ [visited on 2014-03-21] (cit. on p. 45).

Slepak, J., Shivers, O., and Manolios, P. 2014. An Array-Oriented Language with Static Rank Polymorphism. In Programming Languages and Systems. 23rd European Symposium on Programming, ESOP 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014. Ed. by Shao, Z. Lecture Notes in Computer Science 8410. Berlin: Springer, 27–46. DOI: 10.1007/978-3-642-54833-8_3 (cit. on p. 26).

Smith, C. H. and Dunsmore, H. E. 1982. On the relative comprehensibility of various control structures by novice Fortran programmers. International Journal of Man-Machine Studies 17 (2), 165–171. DOI: 10.1016/S0020-7373(82)80017-5 (cit. on pp. 76, 83, 179).

Smith, G. C. S. and Pell, J. P. 2003. Parachute Use to Prevent Death and Major Trauma Related to Gravitational Challenge. Systematic review of randomized controlled trials. BMJ 327 (7429), 1459–1461. DOI: 10.1136/bmj.327.7429.1459. ⟨URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC300808/⟩ (cit. on p. 51).

Soloway, E., Bonar, J., and Ehrlich, K. 1983. Cognitive strategies and looping constructs: an empirical study. Communications of the ACM 26 (11), 853–860. DOI: 10.1145/182.358436 (cit. on pp. 76, 81, 82, 180, 181).

Spinellis, D. 2005. Java Makes Scripting Languages Irrelevant? IEEE Software 22 (3), 70–71. DOI: 10.1109/MS.2005.67 (cit. on p. 24).

SpringerLink s.d. Search FAQ. ⟨URL: http://www.springerlink.com/help/search-tips.mpx⟩ [visited on 2011-10-19] (cit. on p. 61).

St. Pierre, E. A. 2012. Another Postmodern Report on Knowledge. Positivism and its others. International Journal of Leadership in Education. Theory and Practice 15 (4), 483–503. DOI: 10.1080/13603124.2012.696710 (cit. on p. 52).

Steele Jr., G. L. 1999. Growing a Language. Higher-Order and Symbolic Computation 12 (3), 221–236. DOI: 10.1023/A:1010085415024 (cit. on p. 34).

Steele Jr., G. L. 2006. Thoughts on Language Design. Dr. Dobb's Journal 31 (1), 31–32 (cit. on p. 34).

Stefik, A. and Gellenbeck, E. 2011. Empirical studies on programming language stimuli. Software Quality Journal 19 (1), 65–99. DOI: 10.1007/s11219-010-9106-7 (cit. on pp. 71, 76, 84, 88, 89, 180).

Stefik, A., Hanenberg, S., McKenney, M., Andrews, A., Yellanki, S. K., and Siebert, S. 2014. What Is the Foundation of Evidence of Human Factors Decisions in Language Design? An Empirical Study on Programming Language Workshops. To be presented in the 22nd International Conference on Program Comprehension, June 2–3, 2014, in Hyderabad, India (cit. on pp. 36, 38, 93).

Stefik, A. and Siebert, S. 2013. An Empirical Investigation into Programming Language Syntax. ACM Transactions on Computing Education 13 (4), a19. DOI: 10.1145/2534973 (cit. on pp. 13, 37).

Stefik, A., Siebert, S., Stefik, M., and Slattery, K. 2011. An empirical comparison of the accuracy rates of novices using the quorum, perl, and randomo programming languages. In Proceedings of the 3rd ACM SIGPLAN workshop on Evaluation and usability of programming languages and tools. New York, NY, USA: ACM, 3–8. DOI: 10.1145/2089155.2089159 (cit. on pp. 14, 38, 76, 180).

Stern, N. 1979. The History of Programming Languages Conference. I. From an Historian's Perspective. Annals of the History of Computing 1 (1), 68–71. DOI: 10.1109/MAHC.1979.10006 (cit. on p. 34).

Stork, S., Naden, K., Sunshine, J., Mohr, M., Fonseca, A., and Aldrich, J. 2014. ÆMINIUM. A Permission-Based Concurrent-by-Default Programming Language Approach. ACM Transactions on Programming Languages and Systems 36 (1), a2 (cit. on p. 17).

Strachey, C. 2000. Fundamental Concepts in Programming Languages. Higher-Order and Symbolic Computation 13 (1–2), 11–49. DOI: 10.1023/A:1010000313106 (cit. on p. 29).

Straus, S. E., Glasziou, P., Richardson, W. S., and Haynes, R. B. 2011. Evidence-Based Medicine. How to practice and teach it. 4th ed. Edinburgh: Churchill Livingstone (cit. on pp. 14, 38, 39, 43).

Stroustrup, B. 1988. What is Object-Oriented Programming? IEEE Software 5 (3), 10–20. DOI: 10.1109/52.2020 (cit. on p. 23).

Stroustrup, B. 1994. The Design and Evolution of C++. Reading, Massachusetts: Addison-Wesley (cit. on p. 35).

Stroustrup, B. 2014. The C++ Programming Language. 4th ed. Upper Saddle River: Addison-Wesley (cit. on pp. 24, 35).

Stuchlik, A. and Hanenberg, S. 2011. Static vs. dynamic type systems: an empirical study about the relationship between type casts and development time. In Proceedings of the 7th symposium on Dynamic languages. New York, NY, USA: ACM, 97–106. DOI: 10.1145/2047849.2047861 (cit. on pp. 76, 79, 80, 180).

Suri, H. 2013. Epistemological pluralism in research synthesis methods. International Journal of Qualitative Studies in Education 26 (7), 889–911. DOI: 10.1080/09518398.2012.691565 (cit. on p. 53).

Sutherland, W. J., Pullin, A. S., Dolman, P. M., and Knight, T. M. 2004. The need for evidence-based conservation. Trends in Ecology and Evolution 19 (6), 305–308. DOI: 10.1016/j.tree.2004.03.018. [Visited on 2011-10-04] (cit. on p. 38).

Taivalsaari, A. 1993. On the Notion of Object. Journal of Systems and Software 21 (1), 4–16. DOI: 10.1016/0164-1212(93)90013-N (cit. on p. 23).

Taivalsaari, A. 1996. On the Notion of Inheritance. ACM Computing Surveys 28 (3), 438–479. DOI: 10.1145/243439.243441 (cit. on p. 23).

Taveira, J. C., Queiroz, C., Lima, R., Saraiva, J., Soares, S., Oliveira, H., Temudo, N., Araujo, A., Amorim, J., Castor, F., and Barreiros, E. 2009. Assessing Intra-application Exception Handling Reuse with Aspects. In Software Engineering, 2009. SBES '09. XXIII Brazilian Symposium on, 22–31. DOI: 10.1109/SBES.2009.21 (cit. on pp. 76, 180).

Tenny, T. 1985. Procedures and comments vs. the banker's algorithm. SIGCSE Bulletin 17 (3), 44–53. DOI: 10.1145/382208.382523 (cit. on pp. 76, 81, 82, 180).

The Debian Policy Manual 2013. Version 3.9.5.0. Oct. 28, 2013. ⟨URL: https://www.debian.org/doc/debian-policy/policy.pdf.gz⟩ [visited on 2014-04-03] (cit. on p. 149).

The Python Language Reference 2014. ⟨URL: http://docs.python.org/3/reference/⟩ (cit. on p. 18).

Thies, W. and Amarasinghe, S. 2010. An empirical characterization of stream programs and its implications for language and compiler design. In Proceedings of the 19th international conference on Parallel architectures and compilation techniques, 365–376. DOI: 10.1145/1854273.1854319 (cit. on pp. 76, 85, 180).

Thomas, G. and Pring, R., eds. Evidence-Based Practice in Education 2004. Maidenhead, Berkshire: Open University Press (cit. on p. 38).

Tobin-Hochstadt, S. and Felleisen, M. 2008. The design and implementation of typed scheme. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL), 395–406. DOI: 10.1145/1328438.1328486 (cit. on pp. 76, 180).

Tofan, D., Galster, M., Avgeriou, P., and Weyns, D. 2011. Software Engineering Researchers' Attitudes on Case Studies and Experiments. An Exploratory Survey. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 91–95. DOI: 10.1049/ic.2011.0011 (cit. on p. 90).

Tomassetti, F., Rizzo, G., Vetro', A., Ardito, L., Torchiano, M., and Morisio, M. 2011. Linked Data approach for selection process automation in Systematic Reviews. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 31–35. DOI: 10.1049/ic.2011.0004 (cit. on p. 47).

Tonella, P. and Ceccato, M. 2005. Refactoring the aspectizable interfaces: an empirical assessment. Software Engineering, IEEE Transactions on 31 (10), 819–832. DOI: 10.1109/TSE.2005.115 (cit. on pp. 76, 180).

Trancón y Widemann, B. 2009. Church vs. Curry. A Modern Introduction to the Fundamental Dichotomy of Type System Paradigms. In Programmiersprachen und Rechenkonzepte. 26. Workshop der GI-Fachgruppe „Programmiersprachen und Rechenkonzepte". Ed. by Hanus, M. and Braßel, B. Bericht 0915. Institut für Informatik der Christian-Albrechts-Universität zu Kiel, 50–61. ⟨URL: http://www.informatik.uni-kiel.de/uploads/tx_publication/tr_0915.pdf⟩ [visited on 2014-04-22] (cit. on p. 33).

Turner, D. A. 2013. Some History of Functional Programming Languages. (Invited Talk). In Trends in Functional Programming. 13th International Symposium, TFP 2012, St. Andrews, UK, June 12–14, 2012, Revised Selected Papers. Ed. by Loidl, H.-W. and Peña, R. Lecture Notes in Computer Science 7829. Heidelberg: Springer, 1–20. DOI: 10.1007/978-3-642-40447-4_1 (cit. on p. 31).

Turner, M., Kitchenham, B., Budgen, D., and Brereton, P. 2008. Lessons Learnt Undertaking a Large-scale Systematic Literature Review. In Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE). ⟨URL: http://ewic.bcs.org/content/ConWebDoc/19549⟩ [visited on 2014-03-21] (cit. on p. 47).

Turner, R. 2007. Understanding Programming Languages. Minds & Machines 17 (2), 203–216. DOI: 10.1007/s11023-007-9062-6 (cit. on p. 17).

Turner, R. 2009. The Meaning of Programming Languages. Newsletter on Philosophy and Computers 9 (1), 2–7. ⟨URL: http://cswww.essex.ac.uk/staff/turnr/Mypapers/v09n1_Computers.pdf⟩ [visited on 2014-02-10] (cit. on p. 17).

Unger, B. and Prechelt, L. 1998. The impact of inheritance depth on maintenance tasks: Detailed description and evaluation of two experiment replications. Technical report. ⟨URL: http://www.ipd.uka.de/Tichy/uploads/publikationen/143/inheritTR.pdf⟩ (cit. on pp. 76, 79, 80, 179, 181).

UNIVAC FLOW-MATIC Programming System 1958. Remington–Rand Univac (cit. on p. 30).

Wadler, P. 2000. Old ideas form the basis of advancements in functional programming. Dr. Dobb's Journal (Dec. 2000), 37–41. ⟨URL: http://www.drdobbs.com/old-ideas-form-the-basis-of-advancements/184404384⟩ (cit. on p. 33).

Wagenmakers, E. J., Wetzels, R., Borsboom, D., and Maas, H. L. J. van der 2011. Why Psychologists Must Change the Way They Analyze Their Data. The Case Of Psi: Comment on Bem (2011). Journal of Personality and Social Psychology 100 (3), 426–432. DOI: 10.1037/a0022790 (cit. on p. 51).

Vaismoradi, M., Turunen, H., and Bondas, T. 2013. Content analysis and thematic analysis. Implications for conducting a qualitative descriptive study. Nursing and Health Sciences 15 (3), 398–405. DOI: 10.1111/nhs.12048 (cit. on p. 49).

Valente, M., Couto, C., Faria, J., and Soares, S. 2010. On the benefits of quantification in AspectJ systems. Journal of the Brazilian Computer Society 16 (2), 133–146. DOI: 10.1007/s13173-010-0008-0 (cit. on pp. 76, 180).

Walker, R. J., Bamassad, E. L. A., and Murphy, G. C. 1998. Assessing Aspect-Oriented Programming: Preliminary Results. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543, 586. DOI: 10.1007/3-540-49255-0_131 (cit. on pp. 76, 180).

Walker, R. J., Baniassad, E. L. A., and Murphy, G. C. 1999. An initial assessment of aspect-oriented programming. In Proceedings of the 21st international conference on Software engineering. ICSE '99. New York, NY, USA: ACM, 120–130. DOI: 10.1145/302405.302458 (cit. on pp. 76, 180).

Walker, W., Lamere, P., and Kwok, P. 2002. FreeTTS: a performance case study. Tech. rep. TR-2002-114. Sun Microsystems, Inc. ⟨URL: http://labs.oracle.com/techrep/2002/abstract-114.html⟩ (cit. on pp. 76, 180).

Van Roy, P. 2009. Programming Paradigms for Dummies. What Every Programmer Should Know. In New Computational Paradigms for Computer Music. Ed. by Assayag, G. and Gerzso, A. IRCAM / Delatour. ⟨URL: http://www.info.ucl.ac.be/~pvr/VanRoyChapter.pdf⟩ [visited on 2014-04-10] (cit. on pp. 22, 24).

Webster, J. and Watson, R. T. 2002. Analyzing the Past to Prepare for the Future. Writing a Literature Review. MIS Quarterly 26 (2), xiii–xxiii. ⟨URL: http://www.jstor.org/stable/4132319⟩ [visited on 2014-03-14] (cit. on p. 45).

Wegner, P. 1976. Programming Languages. The First 25 Years. IEEE Transactions on Computers C-25 (12), 1207–1225. DOI: 10.1109/TC.1976.1674589 (cit. on p. 34).

Wegner, P. 1987. Dimensions of Object-Based Language Design. In OOPSLA '87. Object-Oriented Programming Systems, Languages and Applications, 168–182. DOI: 10.1145/38765.38823 (cit. on p. 23).

Wegner, P. 1990. Concepts and Paradigms of Object-Oriented Programming. ACM SIGPLAN OOPS Messenger 1 (1), 7–87. DOI: 10.1145/382192.383004 (cit. on p. 21).

Weimer, W. and Necula, G. C. 2008. Exceptional situations and program reliability. ACM Transactions on Programming Languages and Systems 30 (2). DOI: 10.1145/1330017.1330019 (cit. on pp. 76, 85, 180).

Weinberg, G. M. 1971. The Psychology of Computer Programming. New York: Van Nostrand Reinhold (cit. on pp. 13, 36).

Weiner, L. H. 1978. The Roots of Structured Programming. In SIGCSE'78. Papers of the SIGCSE/CSA technical symposium on Computer Science education, 243–254. DOI: 10.1145/990555.990636 (cit. on p. 23).

Vessey, I., Ramesh, V., and Glass, R. L. 2005. A unified classification system for research in the computing disciplines. Information and Software Technology 47 (4), 245–255. DOI: 10.1016/j.infsof.2004.08.006 (cit. on pp. 49, 94, 95, 243).

Vessey, I. and Weber, R. 1984a. Conditional statements and program coding: an experimental evaluation. International Journal of Man-Machine Studies 21 (2), 161–190. DOI: 10.1016/S0020-7373(84)80065-6. ⟨URL: http://www.sciencedirect.com/science/article/pii/S0020737384800656⟩ (cit. on pp. 71, 76, 81, 82, 180).

Vessey, I. and Weber, R. 1984b. Research on Structured Programming: An Empiricist's Evaluation. Software Engineering, IEEE Transactions on SE-10 (4), 397–407. DOI: 10.1109/TSE.1984.5010252 (cit. on pp. 76, 180, 181).

Westbrook, E., Zhao, J., Budimlić, Z., and Sarkar, V. 2012. Practical Permissions for Race-Free Parallelism. In ECOOP 2012 – Object-Oriented Programming. Ed. by Noble, J. Vol. 7313. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 614–639. DOI: 10.1007/978-3-642-31057-7_27 (cit. on pp. 76, 85, 89, 180).

Wexelblat, R. L., ed. History of Programming Languages 1981. ACM monograph series. New York, NY: Academic (cit. on p. 34).

Wheeler, D. J. 1992. The EDSAC Programming Systems. IEEE Annals of the History of Computing 14 (4), 34–40. DOI: 10.1109/85.194053 (cit. on p. 20).

White, G. 2004. The Philosophy of Computer Languages. In The Blackwell Guide to the Philosophy of Computing and Information. Ed. by Floridi, L. Malden, MA: Blackwell, 237–247 (cit. on p. 17).

Whitehead, A. N. and Russell, B. 1910. Principia Mathematica. Vol. I. Cambridge: University Press. ⟨URL: http://name.umdl.umich.edu/AAT3201.0001.001⟩ (cit. on p. 32).

Whiting, P. W. and Pascoe, R. S. V. 1994. A History of Data-Flow Languages. IEEE Annals of the History of Computing 16 (4), 38–59. DOI: 10.1109/85.329757 (cit. on p. 34).

Whitley Jr., B., Kite, M. E., and Adams, H. L. 2013. Principles of Research in Behavioral Science. 3rd ed. New York: Routledge (cit. on pp. 50, 51).

Wiedenbeck, S. and Ramalingam, V. 1999. Novice comprehension of small programs written in the procedural and object-oriented styles. International Journal of Human-Computer Studies 51 (1), 71–87. DOI: 10.1006/ijhc.1999.0269 (cit. on pp. 76, 180, 181).

Wiedenbeck, S., Ramalingam, V., Sarasamma, S., and Corritore, C. 1999. A comparison of the comprehension of object-oriented and procedural programs by novice programmers. Interacting with Computers 11 (3), 255–282. DOI: 10.1016/S0953-5438(98)00029-0 (cit. on pp. 76, 180, 181).

Wijngaarden, A. van, Mailloux, B. J., Peck, J. E. L., Koster, C. H. A., Sintzoff, M., Lindsey, C. H., Meertens, L. G. T., and Fisker, R. G., eds. Revised Report on the Algorithmic Language Algol 68 1976. Berlin: Springer. DOI: 10.1007/978-3-662-39502-8 (cit. on p. 28).

Wilkes, M. V., Wheeler, D. J., and Gill, S. 1951. The Preparation of Programs for an Electronic Digital Computer. With special reference to the EDSAC and the use of a library of subroutines. Cambridge 42, MA: Addison–Wesley (cit. on p. 20).

Wirth, N. 1974. On the Design of Programming Languages. In IFIP Congress 74, 386–393 (cit. on p. 34).

Wirth, N. 1976. Algorithms + Data Structures = Programs. Englewood Cliffs: Prentice–Hall (cit. on p. 18).

Wohlin, C., Runeson, P., Mota Silveira Neto, P. A. da, Engström, E., Carmo Machado, I. do, and Almeida, E. S. de 2013. On the reliability of mapping studies in software engineering. Journal of Systems and Software 86 (10), 2594–2610. DOI: 10.1016/j.jss.2013.04.076 (cit. on pp. 40, 45).

Volos, H., Welc, A., Adl-Tabatabai, A.-R., Shpeisman, T., Tian, X., and Narayanaswamy, R. 2009. NePaLTM: Design and Implementation of Nested Parallelism for Transactional Memory Systems. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653, 123–147. DOI: 10.1007/978-3-642-03013-0_7 (cit. on pp. 70, 76, 180).

Yin, R. K. 2009. Case Study Research. Design and Methods. 4th ed. Los Angeles: Sage (cit. on pp. 49, 90).

Zhang, H. and Ali Babar, M. 2013. Systematic reviews in software engineering. An empirical investigation. Information and Software Technology 55 (7), 1341–1354. DOI: 10.1016/j.infsof.2012.09.008 (cit. on p. 40).

Zhang, H., Ali Babar, M., Bai, X., Li, J., and Huang, L. 2011. An Empirical Assessment of a Systemati Search Process for Systematic Reviews. In Proceedings of the 15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011), 56–65. DOI: 10.1049/ic.2011.0007 (cit. on pp. 44, 49).

Zhang, H., Ali Babar, M., and Tell, P. 2011. Identifying relevant studies in software engineering. Information and Software Technology 53 (6), 625–637. DOI: 10.1016/j.infsof.2010.12.010 (cit. on pp. 44, 45, 49, 62, 63, 94).

APPENDIX 1 RECORD KEEPING AND TOOLS USED

APPENDIX 1.1 Database format

All data collection was recorded in a semistructured, human-readable database consisting of files committed to a Git[1] repository, with commits made generally at least once per day, on those days that data was collected. A public copy of the repository is available at https://yousource.it.jyu.fi/antti-juhani-kaijanaho-s-licentiate-thesis-materials/collected-data.

The use of Git provides several advantages. Most importantly, it preserves data collection history. The sequence of data collection, including any corrections made, is available for examination in that history data. Further, Git uses cryptographical checksums to provide a chain of trust from the most recent version to all historical versions: so long as one trusts that one has the correct current version (and so long one trusts the cryptography in Git), one can equally trust its history as Git reports it. Accidental history editing is completely prevented, and deliberate editing of the history would be reflected in the identity metadata of the current version. The current version that this thesis reports on is identified by `3cd0098c89debac91a0dd9feb26e5dfee95f0fc8`.

APPENDIX 1.1.1 General syntax

The general format of most files in the database resembles the format of the header of an Internet mail message (Pogran et al. 1977; Crocker 1982; Resnick 2008) and the format used in many Debian package control files (*The Debian Policy Manual* 2013, Section 5). I originally chose this format in order to be able to use my `dctrl-tools` toolset,[2] which was designed to handle the Debian control file format. This format is also human-readable and easy to write using a text editor, which is a significant advantage.

Every line in a file conforming to this syntax is either a *field-beginning line*, a *field-continuation line* or an *empty line*. Every field-beginning line starts with a non-whitespace character and contains at least one colon; every field-continuation line starts with a whitespace character and contains at least one non-whitespace character.

It is a syntax error for any line to consist solely of whitespace, and for a line starting with a non-whitespace character to lack a colon. It is also a syntax error for a field-continuation line to occur anywhere except immediately after a field-beginning line or after another field-continuation line.

A *field* starts with a field-beginning line and contains all immediately following field-continuation lines. The *name* of the field consists of everything on the field-beginning line up to and not including the first colon on the line. The *content* of the field starts with the first non-whitespace character following the first colon

1 Git is a version control software suite, originally developed by Linus Torvalds. Its principal web site is http://git-scm.com/.

2 https://packages.debian.org/unstable/dctrl-tools

on the field-beginning line, and extends to the last character of the last line of the field, not including the final line terminator character, if any. Any period that occurs as the first non-whitespace character of a field-continuation line is ignored for the purposes of field content; this special treatment, sometimes called *dot-stuffing*, allows a field to include empty lines (represented as field-continuation lines consisting of a single period).

A *record* begins with a field-beginning line that either is the first line of a file or directly follows an empty line. It ends with the first subsequent field-beginning or field-continuation line that is immediately followed by either an empty line or the end of the file.

APPENDIX 1.1.2 Search records

Each search performed in this mapping study, with one exception, was recorded as a file in the subdirectory `searches/`. Only the snowball search has no separate record. Each file was named with a short tag indicating the search venue followed by a dash and an ordinal number distinguishing between multiple searches in the same venue. For example, the file `searches/popl-4` records the fourth manual search in the Proceedings archive for the ACM SIGACT–SIGPLAN Symposia on Principles of Programming Languages Proceedings (POPL). That file has the following content:

```
Search: popl-4
Protocol-Version: 3
Date: 2011-09-01
Description: Manual scan of the ASM SIGACT-SIGPLAN Symposium on
 Principles of Programming Languages (POPL) proceedings for the years
 2008-2011, as recorded at ACM Digital Library. Any work that might
 satisfy one of the following criteria is recorded for further study:
  1. The work is a primary study that attempts to determine the efficacy
     of a programming language design decision.
  2. The work is a literature review that attempts to summarise or
     consolidate research on the efficacy of a programming language
     design decision.
 The formal level of scrutiny is that "obviously fails both" implies
 "do not record".  In practice, when I believe a work fails both
 criteria, I still record it if I feel I need to explain the decision
 (which I leave to Phase Two).
Metadata-Used: Title and author; abstract and keywords if available and
 needed to resolve undecidedness.
```

These files contain exactly one record comprising the following fields:

Search: the name of the file

Protocol-Version: the version number of the study protocol in force at the time of the search

Date: the dates of the search, in the International Standard format YYYY-MM-DD

Description: a verbal description of the search, including the venue searched; I also often paraphrased the selection criteria as they were applicable in the search phase

Metadata-Used: a verbal description of the metadata that were used to evaluate selection criteria during the search; unfortunately, this field did not always accurately describe the actual practice

SearchTerm: for automatic searches, the search term verbatim, as input to the search engine; for example, `gs-9` contains

```
SearchTerm:  "programming language" intitle:"case study",
```

all in one line

ReportedHitCount: for automatic searches, the number of hits reported by the search engine for this particular search

YearRestriction: for automatic searches, any restriction given to the search engine on which years to search

X-ISI: used only in the Web of Science searches, to record the search parameters verbatim; for example, `ws-2` contains `X-ISI: Databases=SCI-EXPANDED, SSCI, A&HCI Timespan=2011-01-01 - 2012-12-01 Lemmatization=On`, in two lines

Problem: used only once (in `gs-2`), to record a particular serious problem encountered in the search: `PROBLEM: GOOGLE SCHOLAR REFUSES TO SHOW HITS BEYOND THE FIRST THOUSAND!`

APPENDIX 1.1.3 Publication records

I recorded every publication I found in a search and did not regard as obviously out of scope. Each publication has its own file in the `refs/` subdirectory, named by the surname of the first author, followed by a dash and the publication year, followed by a disambiguating letter, if necessary, and finally followed by the filename extension `.txt`. The name of the file, without the filename extension, is used as a cross-reference identifier of the publication described by the file. For example, the file `refs/hanenberg-2010a.txt` describes a publication whose cross-reference identifier is `hanenberg-2010a` and contains the following:

```
Authors: Stefan Hanenberg
Title: Doubts about the Positive Impact of Static Type Systems on Programming
 Tasks in Single Developer Projects - An Empirical Study
Booktitle: Proc. ECOOP 2010 European Conference on Object-Oriented Programming
Series: Lecture Notes in Computer Science
Number: 6183
Pages: 300-303
Year: 2010
DOI: 10.1007/978-3-642-14107-2_14
SelectionDecision: II.ajk/2011-11-23 PASSED primary:YES litrev:NO pub:YES
 lang:YES priEvidence:YES
SelectionDecision: III.ajk/2012-12-04 PASSED primary:YES litrev:NO pub:YES
 lang:YES priEvidence:YES; This short article reports a controlled experiment
```

```
 with human participants evaluating the efficacy of dynamic/static type
 systems.
Searches: ecoop-15/2011-08-19 gs-2/2011-09-07 gs-5/2011-09-13
 cites:gannon-1977/2013-02-19 citedby:hanenberg-2010/2013-02-21
 citedby:kleinschmager-2012a/2013-02-22 cites:prechelt-1998/2013-03-05
 citedby:stuchlik-2011/2013-03-11
Snowballed: first.baddate/2013-04-11
SelectionDecision: final/2013-08-07 PASSED priEvidence:YES primary:YES
 litrev:NO lang:YES pub:YES; [III.ajk]This short article reports a controlled
 experiment
 with human participants evaluating the efficacy of dynamic/static type
 systems.
```

These files contain exactly one record. All publication records could use the following bibliographical fields, which are mostly self-explanatory: Authors (in BibTEX format), Title, Year, Pages, DOI, URL, URI, and Month. Journal publications used the following additional fields: Journal, Volume, Number, ISSN, and Articleno (for journals that uses article numbers instead of, or in addition to, page numbering). Books used the following additional fields: ISBN, ISSN, Publisher, Address (for the publisher's location), Series, Volume, and Number. Publications that are a part of some book use, in addition to the book fields, the following fields: Editor, Booktitle, Location (for conference proceedings), and Articleno. Theses used the additional fields School and ThesisType. Departmental reports used the additional fields Institution, ReportType and ReportID.

All publications also made use of the following process-related fields:

SelectionDecision: describes a particular selection decision made with regard to this publication, in the following manner:

- First, there is a tag identifying the phase in which the decision was made (most commonly "II", for initial decisions made based on metadata only, "III" for initial decisions based on full text, and "final" for the final decision) and often also the decision-maker (affixing a period and their nickname, most commonly ".ajk" indicating myself, to the tag).
- Then, there is a solidus followed by the date of the decision in the International Standard format.
- Then, there is whitespace followed by an indication of the bottom-line decision made, either "PASSED" (the publication was not excluded) or "EXCLUDED".
- Then, answers to the selection criterion questions are listed, separated by whitespace. Each answer is encoded by a tag naming the question (Q1 is `primary`, Q2 is `litrev`, Q3 is `pub`, Q4 is `lang`, Q5 is `priEvidence`, Q6 is `secContainsEvidence`, and Q7 is `secDiscussEvidence`) followed by a colon and either "YES" or "NO".
- Finally, and optionally, the answers may be followed by a semicolon followed by a free-form explanation of the decision.

Sequence: indicates that this publication belongs in a named sequence and has a particular ordinal number within that sequence; the sequence name and the ordinal are separated by a colon; for example `youngs-1974.txt` contains `Sequence: selection-control-sample:92`, on one line

Searches: indicates, the searches that located this publication; note that for a borderline publication, the list may not be complete as in some searches I may have decided that the publication obviously is off topic even though I declined to make that decision in other searches; the searches are listed in an arbitrary order and separated by whitespace, encoding each search as follows:

- First, the search is named. The name is either a file name in the `searches/` directory, or a snowballing reference. The latter start with either `cites:` or `citedby:` followed by the cross-reference identifier of the publication that this publication cites or that this publication is cited by, respectively.
- Then, a solidus separates the search name from the date.
- Finally, the date on which this publication was found by the search is given in the International Standard format.

Snowballed: contains, if present, a tag and a date indicating that the publication has been subjected to snowballing. The idea was to indicate which round of snowballing the publication was subjected to, but since there was only one round, and because that round was conducted before I thought of recording this information, the tag and date are (beyond their mere presence) meaningless.

SelectionNote: contains a free-form textual note made during the selection process (for example, I commonly used it to indicate that an interlibrary loan request was pending)

Forum: contains an identifier for the publication forum (one identifier for each journal, conference proceedings series etc)

Disregarded-SelectionDecision: was used in rare cases to retain a record of a selection decision that was overridden later

APPENDIX 1.1.4 Study records

I assigned a study identifier, in the form of S*n*, where *n* is a positive integer, to each publication finally selected for inclusion. Most publications received one unique identifier. If two publications reported the same study, they received the same identifier. If a publication reported more than one study, it may or may not have received more than one identifier (typically not, but for example if one of the studies was reported in another publication as well but another was not, assigning more than one identifier to the publication was necessary).

For each study identifier, I created a file named after the identifier in the subdirectory `extracts/`, with the filename extension `.txt`. The directory name originates from that the files were at first only used to record extracted data from each study; it became a misnomer later, but renaming the directory would cause more hassle than correcting such a cosmetic issue is worth. For example, the file `extracts/S135.txt` contains the following:

```
Study: S135
Articles: shneiderman-1975
Method: "the focus of this paper is on experiments in programming
 language features, stylistic considerations and design techniques."
 (p. 653)
Codes: NarrativeReview

DesignChoice: "the IF-THEN-ELSE construction and the IF(CONDITION)GOTO
 statement" (p. 654)
Efficacy: "easier to use and resulted in fewer bugs" (p. 654)
Method: "a fascinating experiment on non-programmers [...] based on a
 relatively small sample size in a carefully controlled, but
 artificial programming environment" (p. 654)
IdentifiedResults: "the IF-THEN-ELSE construction was easier to use and resulted
 in fewer bugs, particularly with more complex problems."
PriorStudies: not indicated
IncludedStudies: sime-1973
Codes: FeatureDesign Conditionals
       ProgrammingEffort ErrorProneness
       NonrandomizedControlledExperiment
       NonProgrammers
```

Each such file has at least one record, and most have at least two. The first record contains metadata about the study identifier; the subsequent records each document one distinct primary study reported in the publications assigned to this identifier. To distinguish between studies as identified by the S*n* identifiers and studies as described by these records, the latter are called *sub-studies* (there being usually one or more sub-study in each study).

The first record of each file in the `extracts/` directory uses the following fields:

Study: contains the study identifier of this study

Articles: lists each publication (identified by their cross-reference identifiers and separated by whitespace) that this study identifier is assigned to

Note: contains a free-form textual note

Method: occurs only in secondary studies and carries a free-form text (usually an attributed direct quote) describing the secondary-study method used

Codes: occurs only in secondary studies, and contains secondary-study method codes (see Table 22 on page 175), separated by whitespace, assigned to this study

Exclude: contains, if present, either "NO" or "YES", the latter indicating that the study this file describes was excluded from this mapping study after study identifiers were assigned; such exclusions are referred to as being *post hoc*.

ExclusionReason: is used only if Exclude: YES is present and contains a free-form explanation of the exclusion

Any study that was not post-hoc excluded will have at least one sub-study. For primary studies, the sub-studies are usually self-contained but related separate studies that have been reported in the same publication. For secondary studies, the sub-studies are the primary studies discussed by the secondary study, as they are described by the secondary study.

All sub-studies can make use of the following fields:

DesignChoice: contains free-form text (usually an attributed direct quote) describing the design decisions whose relative efficacy was under study

Efficacy: contains free-form text (usually an attributed direct quote) describing the facets of efficacy used in the sub-study

Method: contains free-form text (usually an attributed direct quote) describing the research method used in the sub-study

IdentifiedResults: contains free-form text (usually an attributed direct quote) describing the result of the study as relevant to the question of the relative efficacy of the design decisions studied and as reported in the publications this study identifier was assigned to

PriorStudies: contains free-form text (sometimes an attributed direct quote with bibliographical information) identifying prior studies of relevance to this sub-study, particularly studies this sub-study replicated or followed up on

FollowupTo: contains a whitespace-separated list of publications in this database, identified by their cross-reference identifiers, that this sub-study followed up on

Replicates: contains a whitespace-separated list of publications in this database, identified by their cross-reference identifiers, that this sub-study replicates

OtherPriorStudies: contains a whitespace-separated list of publications in this database, identified by their cross-reference identifiers, that are relevant prior studies to this sub-study in some way other than being followed up on or being replicated.

Note: contains a free-form note

Codes: contain the design-decision (see Table 19 starting on page 169), efficacy (see Table 20 starting on page 173), and primary-study method (see Table 21 starting on page 174) codes, separated by whitespace, assigned to this sub-study

Sub-studies of secondary studies use the following additional fields:

IncludedStudies: contains a whitespace-separated list of publications in this database, identified by their cross-reference identifiers, that the secondary study report cites as reporting the (primary) sub-study

OtherIncludedStudy: contains a free-form bibliography entry for a publication not in this database that the secondary study report cites as reporting the (primary) sub-study; this field is repeated for each such publication

APPENDIX 1.1.5 Codes and raw themes

All codes assigned to studies are defined in the top-level directory file `codes.txt` that contains one record for each code. These records use the following fields:

Code: the name of the code
Category: the category of the code (DesignDecision, Efficacy, Method, or SecondaryMethod)
SubCategory: the subcategory, if any, assigned to the code
Implies: a whitespace-separated list of codes, if any, implied by this code
Definition: a free-form definition of this code

The tables in Appendix 2 have been automatically generated from this file. The following is one record in the file, given as an example:

```
Code: TypeInference
Implies: StaticTyping
Category: DesignDecision
SubCategory: Typing
Definition: The design decisions under study involve type inference
 (that is, static typing where the type system infers the vast majority of
 types that one would normally expect to declare explicitly).
```

In theme development, I have defined a number of parametrized raw themes in the top-level directory file `themes.txt`. The `slr-tools` toolset can be used to generate lists of studies belonging to each raw theme and the parameter values associated with each study. Each raw theme is described by a single record in the file, using the following fields:

Theme: the name of this raw theme
Query: a term in the query language (Section 1.2) describing the set of studies belonging in this raw theme and any raw-theme parameters associated with each such study

The following is one record in the file, given as an example:

```
Theme: FeatureDesignOrPresence
Query:
 such study in studies and
      @ss in study.subStudies and
      feature in ss.codes
 that feature.category.name == DesignDecision &&
      feature.name != IndirectRelevance &&
      feature.subcategory.name != Paradigm &&
      feature.subcategory.name != SpecificLanguage &&
      feature.subcategory.name != DecisionType &&
      exists such c in ss.codes that (c.name == FeatureDesign
                                  || c.name == FeaturePresence)
```

APPENDIX 1.2 The query language

The syntax of the query language used in raw themes and with the `slr-tools query` tool, which is introduced later, is specified using the following context-free grammar metalanguage:[3] terminal symbols that stand for more than one lexeme as well as nonterminal symbols are written in *italics*, terminals that stand for themselves are written in **`fixed-width bold`**; a set of productions starts with a nonterminal followed by a colon, with the right-hand sides listed below, indented, each on its own line. As an abbreviation, a symbol may be given the subscript "opt", meaning that the symbol is optional. Similarly, if all productions of a particular nonterminal have one-symbol right-hand-sides, those symbols may be listed on one line, if the colon is followed by "one of".

The starting symbol of the grammar is *term*.

APPENDIX 1.2.1 Lexical structure

I will employ the conventional two-level language structure in the following language description. The valid expressions of the query language are strings of Unicode characters. The lower (lexical) layer of the language assigns every non-whitespace character in a syntactically valid expression into exactly one *lexeme*. Lexemes are contiguous substrings of the expression that do not overlap and that do not contain any whitespace. Every lexeme is associated with exactly one terminal symbol of the context-free grammar specifying the upper (syntactic) layer of the language. The grammar sees the expression as a sequence of terminal symbols and does not care what the underlying lexemes are; but, of course, the particular lexeme underlying a particular terminal symbol has semantic significance.

Most lexemes in this language are *words*, which come in two forms. Simple words start with a Unicode identifier-start character and continue with Unicode identifier-part characters. Quoted words start with a **`"`** or a **`'`**. Within a quoted word, backslashes are ignored, except those backslashes that themselves are preceded by an ignored backslash. A quoted word ends with the first occurrence of its starting character that is not preceded by an ignored backslash. The starting and ending character are not included in the quoted word; thus **`"foo"`** and **`foo`** are the same word.

The following are the only lexemes in the language that are not words:

`( ) = @ .  , && || => !  == !=`

In the grammar, the symbol *word* is a nonterminal defined as follows:

word:
  keyword
  other-word

[3] This metalanguage is substantially similar to that used by *Information Technology – Programming Languages – C#* (2006), Gosling et al. (2014), and *Information Technology – Programming Languages – C* (2011) to specify C#, Java, and C syntax, respectively, among others. The earliest use of it I am aware of is by Ritchie (1974).

keyword: one of
and by count exists group in of powerset such that

The symbol *other-word* is a terminal symbol whose lexemes comprise those words that are not *keywords*. Note that *keywords* are not reserved words; in contexts where no ambiguity arises, a *keyword* can be used like any *other-word*. Any ambiguities caused by having a particular *keyword* be a *word* are resolved by temporarily treating that *keyword* as if it were not a *word*.

APPENDIX 1.2.2 Terms

term:
powerset-term
groupby-term
countin-term
postfix-term

postfix-term:
projection-term
primary-term

primary-term:
query-term
variable
(*term* **)**

Terms are expressions that evaluate into values.

variable:
word

Variables are either free or bound. A bound variable is one that has been given a value somewhere in the lexical environment. A bound variable, used as a term, evaluates to its bound value. All variables that are not bound are free, and evaluate to string values that represent the variables themselves; thus, if for example **foo** is a free variable, it evaluates to the string value `foo`.

query-term:
such *iterators* **that** *predicate*

iterators:
iterator
iterators **and** *iterator*

iterator:
$\textit{qualifier}_{\text{opt}}$ *variable* **in** term
$\textit{qualifier}_{\text{opt}}$ *variable* **=** term

Query terms are basically list comprehensions, and serve the same function as looping constructs in many languages.

Consider first a query term in which there is exactly one iterator, the iterator is an **in**-iterator, and the iterator has no qualifier. The term in the iterator is first evaluated; it must evaluate into a sequence value. The query term evaluates the predicate for each value in the sequence, with the value bound to the iterator's variable within the predicate. The query term evaluates into a sequence value which contains exactly those values from the iterator's sequence that make the predicate come out true. Thus, this special case query term acts essentially as a sequence filter.

If a query term has at least two iterators, the behavior is different in three main respects. First, each iterator variable is bound not just within the predicate but also within the terms of each of the iterators on its right-hand side within the same query term. Second, all possible bindings of iterator variables to values from their respective iterator-term evaluated sequences are tested against the predicate. Third, the resulting sequence comprises records containing a field for each variable, the sequence recording the variable bindings that made the predicate come out true.

If an iterator is a **=**-iterator, the iterator's term does not need to evaluate to a sequence. Its value is treated as if it were the single element of a sequence value in an otherwise equivalent **in**-iterator.

qualifier:
  @

If an iterator is qualified by an **@**, that particular iterator's variable will not have a corresponding field in the records comprising the sequence generated by the query term. At least one iterator in a query term must lack an **@**-qualifier. An **@**-qualifier makes no sense in a query term containing exactly one iterator, and such usage is prohibited.

projection-term:
  postfix-term **.** *word*

A *projection-term* first evaluates its *postfix-term*. If the *postfix-term* evaluates to a record containing a field named by the *word*, the *projection-term* evaluates to the value carried by that field in the record. If the *postfix-term* evaluates to a sequence, the *projection-term* evaluates to a sequence of the same length, each element of the latter sequence containing the value carried by the field named by the *word* in the record that is the corresponding element of the former sequence, or a null value, if the corresponding element is not a record that contains such a field.

countin-term:
  count *word* **in** *term*

A *countin-term* first evaluates its *term*. It is an error for the value of the *term* not to be a sequence of records, each containing a field named by the *word* that carries a sequence value. The *countin-term* evaluates to a sequence of the same length, each element of which is a copy of the the corresponding record in the original sequence, with the sequence carried in the *word* field replaced by the count of its elements.

groupby-term:
  `group by` *wordlist term*

wordlist:
  word
  wordlist `,` *word*

A *groupby-term* first evaluates its *term*. The value of the term must be a collection of records. The *groupby-term* evaluates to a (usually shorter) collection of records. The latter collection contains one record for each unique combination of values carried by the *wordlist*-named fields in the former collection of records. Each such record carries in each of its other fields a list of all values carried by that field in all the records of the former collection that share the same combination of values in the *wordlist*-named fields.

powerset-term:
  `powerset of` *term*

A *powerset-term* first evaluates its *term*. The value of the term must be a collection. The *powerset-term* evaluates to a collection of all possible subcollections of that collection.

APPENDIX 1.2.3 Predicates

predicate:
  basic-predicate `&&` *predicate*
  basic-predicate `||` *predicate*
  basic-predicate `=>` *predicate*
  basic-predicate

basic-predicate:
  `! (` *predicate* `)`
  `(` *predicate* `)`
  `exists` *term*
  term `==` *term*
  term `!=` *term*
  term `<` *term*
  term `in` *term*

Predicates are truth-valued expressions, which can be built using the usual short-circuiting connectives (`&&` and `||`), an implication connective (with P_1`=>`P_2 defined as syntactic sugar for `!(`P_1`)||`P_2), and the usual negation (with mandatory parentheses) from a number of atomary predicates.

The existence predicate `exists` *term* evaluates the *term*, which must evaluate to a collection value, and comes out true if the collection is nonempty (the name is based on the expectation that the term is often a *query-term*, yielding the phrase `exists such`… `that`). The membership predicate evaluates both terms, requires that the latter evaluates to a collection value, and comes out true if the former evaluates to a member of the collection. The equality and inequality predicates are self-explanatory. The less-than predicate requires that the `term`s are string values and comes out true if `java.lang.String.compareTo` returns a negative number for the string values in question.

APPENDIX 1.2.4 Pre-bound variables

The following variables are pre-bound and can be used in any query, if the corresponding information exists in the database:

studies: containing a collection of records, each describing a study
primaryStudies: containing a collection of records, each describing a study, limited to primary studies
secondaryStudies: containing a collection of records, each describing a study, limited to secondary studies
codes: containing a collection of records, each describing a code (from `codes.txt`)
articles: containing a collection of records, each describing a publication (as described by a file in the `refs/` subdirectory)
includedArticles: containing a collection of records, each describing a publication (as described by a file in the `refs/` subdirectory), restricted to those with a final inclusion decision
searches: containing a collection of records, each describing a search

A record describing a study has the following fields:

name: a string value containing the S*n* study id
secondary: a string value containing either "yes" or "no", indicating whether this is a secondary study
headCodes: a collection of records, each describing a code, as listed in the Codes field of the first record in the study file; these are the secondary study codes, if any
subStudies: a collection of records, each describing a sub-study
articles: a collection of records, each describing a publication, as listed in the Articles field of the first record in the study file
citation: a LaTeX `\cite` command referring to the publications that have been assigned this study identifier
pubYear: a string value containing the year of the earliest publication associated with this study

A record describing a publication has the following fields:

id: a string value containing the cross-reference identifier of the publication
studies: a collection of records, each describing a study, listing the studies that this publication has been assigned to
searches: a collection of records, each describing a search, listing the searches that found this study
included: a string value, either "yes" or "no", indicating whether this study is finally included or not
forum: a string value, taken from the Forum field
year: a string value, taken from the Year field

A record describing a sub-study has the following fields:

id: a string value containing S*na*, where S*n* is the identifier of the study that this sub-study is a part of, and *a* is a letter indicating the ordinal number of this sub-study within the study (missing if the study is a primary study and has exactly one sub-study)

explicitCodes: a collection records, each describing a code, listing (with order preserved) the codes that are given in the Codes field of the sub-study record in the study file, without listing any implied codes unless they are explicitly given in the Codes field

codes: a collection of records, each describing a code, listing all codes, even implied codes, assigned to this sub-study

followupTo: a collection of records, each describing a publication that this study follows up on

replicates: a collection of records, each describing a publication that this study replicates

otherPriors: a collection of records, each describing a publication that is otherwise a significant prior publication to this study

included: a collection of records, each describing a publication that has been selected for inclusion in this mapping study and has been cited by this sub-study

includedCites: a collection of string values, each containing a LaTeX citation to a publication, listing each publication that has been selected for inclusion in this mapping study and has been cited by this sub-study

includedOthers: a collection of string values, each containing a LaTeX citation to a publication, listing each publication that has been excluded from this mapping study and has been cited by this sub-study

A record describing a code has the following fields:

name: a string value containing the name of the code

category: a record value describing a code category

subcategory: a record value describing a code subcategory

implies: a collection of records, each describing a code, listing those codes that this code directly or indirectly implies

definition: a string value containing the definition of the code

A record describing a code category has the following fields:

name: a string value containing the name of the category

subcategories: a collection of records, each describing a subcategory of this category

A record describing a code subcategory has the following fields:

name: a string value containing the name of the category

category: a records describing the category that this subcategory belongs to

codes: a collection of records, each describing a code, listing those codes that belong to this subcategory

A record describing a search has the following fields:

type: a string value, either “automatic”, “manual”, “cites”, or “citedby”, indicating the type of the search
name: (only if type is “automatic” or “manual”) a string value containing the name of the search
baseName: (only if type is “automatic” or “manual”) a string value containing the name of the search, excluding the last dash it contains and everything after it (if any)
key: (only if type is “cites” or “citedby”) a string value containing the cross-reference identifier of the publication that this search snowballed

APPENDIX 1.3 The `slr-tools` toolset

During the course of this mapping study, I wrote a set of programs, in Java, to automate parts of the study. [4] I call this toolset `slr-tools` mainly for convenience; I am aware that the name is not particularly original or unique. It should be noted that these tools are specific to this study, and they cannot be used without changes on other studies. Note also that more recent versions of the toolset do not work with early versions of the database due to format changes; one should generally use the same vintage of both the toolset and the database. The toolset is written in Java 7 and comprises about 8.500 lines of code (counted using David A. Wheeler’s ‘SLOCCount’).[5]

APPENDIX 1.3.1 General usage

There is one entry point, the `fi.jyu.antkaij.SlrTools.main` method. Each tool is called up by giving particular command-line arguments to this entry point. The top-level directory of the database (containing such files as `codes.txt` and `themes.txt`) is assumed to be the current directory, unless another is indicated by giving `-dir` *`DIR`* as the first two command-line arguments to the entry point.

In the subsequent examples, I will be calling this entry point using the following Unix shell script with the name `slr-tools`:

```
#!/bin/sh

exec java -ea -cp /home/ajk/research/mapping-tools/tools/class:$CLASSPATH \
  fi.jyu.antkaij.SlrTools "$@"
```

This script assumes, of course, that the toolset classes have been compiled into the directory `/home/ajk/research/mapping-tools/tools/class`.

The `validate` tool, invoked as `slr-tools validate`, reads the database, checks all files for syntax errors, and performs a number of cross-checks between

4 It is publicly available at https://yousource.it.jyu.fi/antti-juhani-kaijanaho-s-licentiate-thesis-materials/slr-tools.
5 http://www.dwheeler.com/sloccount/

the files (such as that all publications cross-referenced actually have a descriptive file in the `refs/` subdirectory). Many of the other tools perform these checks as well before performing their main function.

APPENDIX 1.3.2 Recording search results

New publications can be added to the database by hand, or by using

```
slr-tools import SEARCHNAME BIB-URI
```

where *SEARCHNAME* is the name of an existing search recorded in the `searches/` directory or, in the case of snowballing, a valid cross-reference identifier of an existing publication preceded by `cites:` or `citedby:`; and *BIB-URI* is a local file name or an URI for a BibTEX record for the publication to add; the standard input (`System.in` in Java terminology) can be indicated by giving `-` as the *BIB-URI*. The import tool will try to determine if the publication already exists in the database, and if so, only amends its `Searches` field. If no existing record is found, the import tool converts the BibTEX record into a record suitable for a file in the `refs` subdirectory, and opens a Swing-based text editor allowing the user to edit the record. Once the user is satisfied, they can ask it to be saved, in which case the import tool checks that the record does not cause validation errors and saves it. If there were validation errors, those are pointed out to the user and they are given an opportunity to re-edit the record.

If a publication is known to exist in the database in the file *REF-FILE*, a new search (named *SEARCHNAME*) can be added to its `Searches` field by using

```
slr-tools add-search SEARCHNAME REF-FILE
```

A `Snowballed` field carrying a particular *TAG* can be added to a particular publication's record in the file *REF-FILE* by using

```
slr-tools add-snowball TAG REF-FILE
```

APPENDIX 1.3.3 Selection decisions

A tag must be given for each selection phase so that `slr-tools` can keep track of which publications have been selected and which have not. In this mapping study, the main tags I have used are `II.ajk`, indicating a main selection round based on metadata only, and `III.ajk`, indicating a main selection round based on the full-text of the publications. The special tag `final` indicates the final decision. I have used other tags to record validation exercises.

If a selection phase tagged *SELECTOR-ID* should go through all recorded publications, ordinarily one would begin a day's work of selection by invoking

```
slr-tools select SELECTOR-ID
```

If it should go through only those publications that have been `PASSED` in an earlier round tagged *FILTER-ID*, the ordinary invocation would be

```
slr-tools select SELECTOR-ID FILTER-ID
```

The *FILTER-ID* can also be a `Sequence` tag preceded by `seq:`. If one wants to record a decision regarding a specific set of publications (the `refs/`-subdirectory file names of which are *REF-FILE*s, the ordinary invocation would be

```
slr-tools select-this SELECTOR-ID REF-FILE...
```

In all cases, the select tool creates an internal set of publications that need to be given a *SELECTOR-ID* selection decision, picks one at random, and then presents the user with a Swing form, like the one depicted in Figure 12, allowing the user

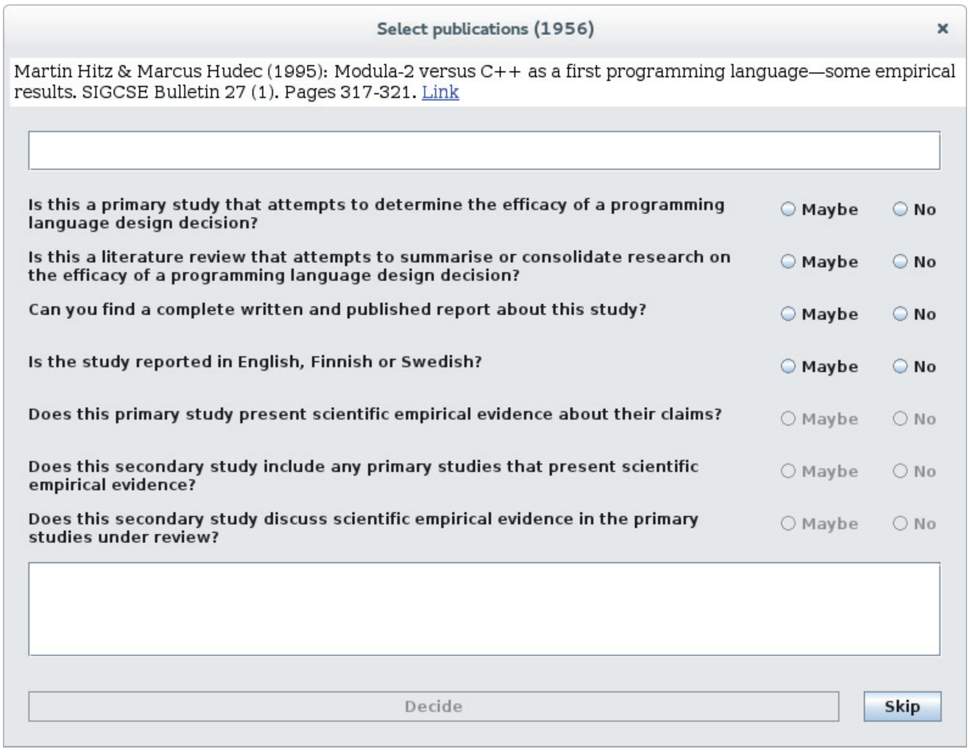

FIGURE 12 The `slr-tools select` form

to answer the selection criteria questions. Some of the questions are disabled initially or in response to answers given to other questions, if the questions do not make sense in light of the other answers given. The "Decide" button is disabled as long as the questions do not dictate a bottom-line decision. Once they determine a decision, the button is enabled and its text changed to "Pass" or "Exclude", based on the decision determined by the current answers. The form also contains two text boxes: the upper one is linked to the publication's `SelectionNote` field, and the latter will be used as the free-form explanation for the selection decision that will be entered using the form. Once a decision has been made, or the "Skip" button pressed, the publication is removed from the internal set and another randomly selected publication from the internal set is presented for decision.

All selection decisions under a tag *OLD-TAG* can be copied over to another tag *NEW-TAG* by invoking

```
slr-tools copy-selection OLD-TAG NEW-TAG
```

APPENDIX 1.3.4 Selection evaluation support

A random sample of `N` publications, selected from those publications that have been `PASSED` in the selection round `FILTER-ID`, can be created using

```
slr-tools random-sample TAG N FILTER-ID
```

This lists the sample, in a random order, to standard output, and also saves the sample as a `Sequence` under the name `TAG`. For example:

```
$ slr-tools random-sample example 10 final
Sample size 10, population 180
   0. hertz-2005
   1. dolado-2003
   2. bartsch-2008
   3. gannon-1975a
   4. walker-1999
   5. gannon-1975
   6. nanz-2010
   7. walker-1998
   8. iselin-1988
   9. qi-2010
$ grep Sequence refs/iselin-1988.txt
Sequence: example:8
$
```

The `FILTER-ID` may be omitted, in which case the sample is created from the full population of all publications in the database.

The Cohen kappa statistic of strength of agreement between two selection rounds `SEL-ID-1` and `SEL-ID-2`, restricted to those publications for which both rounds have a decision and restricted to comparing only the bottom-line decisions, can be computed using

```
slr-tools kappa SEL-ID-1 SEL-ID-2
```

A similar Fleiss kappa statistic between an arbitrary number of selection rounds named by `SEL-ID`s, again similarly restricted, can be computed using

```
slr-tools fleiss-kappa SEL-ID...
```

A CSV-format table of bottom-line decisions of two or more `SEL-ID`s can be obtained using

```
slr-tools ratings-csv SEL-ID...
```

APPENDIX 1.3.5 Report dumps

A number of data dumps, for report-generation purposes, can be generated. The command

```
slr-tools code-frequencies
```

reports the frequency of assignment to studies for each of the defined codes.

The command

```
slr-tools export-html DIR
```

writes to the *DIR* directory a set of static web pages describing most of the database, including pages detailing codes and their assignments, as well as raw themes and their contents.

The command

```
slr-tools dump DECISION SEL-ID
```

generates a LaTeX list (without the enclosing begin and end commands) of all publications for which a particular bottom-line *DECISION* (either `PASSED` or `EXCLUDED`) has been made in the selection round *SEL-ID*. For example, the list in Appendix 5 was generated using the command

```
slr-tools dump EXCLUDED final
```

The command

```
slr-tools dump-bib DECISION SEL-ID
```

generates a BibTeX database of all publications for which a particular bottom-line *DECISION* (either `PASSED` or `EXCLUDED`) has been made in the selection round *SEL-ID*. For example, the BibTeX database of all included studies, which is used in this thesis, was generated using the command

```
slr-tools dump-bib PASSED final
```

The command

```
slr-tools dump-studies N
```

generates content usable in a LaTeX tabular environment that contains all studies that have an assigned S*n* identifier and their `dump-bib` compatible citations in *N* columns. The main content of Table 9 was generated using

```
slr-tools dump-studies 3
```

The command

```
slr-tools dump-sample FMT SEQ
```

can be used to generate a bibliography of the publications in the sequence *SEQ* in the format *FMT* (which is either `text`, `LaTeX` or `HTML`).

APPENDIX 1.3.6 Queries

Finally, `slr-tools` supports queries using the language described above in Appendix 1.2. Queries can be executed on the command line using

```
slr-tools query QUERY-TERM
```

which outputs the query in a human-readable but unformatted style,

```
slr-tools query -table QUERY-TERM
```

which outputs a fixed-width-font textual table of the results,

```
slr-tools query -table=COLUMN,...  QUERY-TERM
```

which outputs a similar table but with the specified *COLUMN*s in the specified order, or

```
slr-tools query -latex=COLUMN,...  QUERY-TERM
```

which outputs the result in a format usable in a LaTeX tabular environment, with the specified *COLUMN*s in the specified order. The *QUERY-TERM*s above are syntactically *term*s, not necessarily *query-term*s.

For a complicated example, the meat of Table 19 was generated using

```
slr-tools query -latex=study,cite,ddcodes,effcodes,metcodes \
  'such @st in primaryStudies and
        @ss in st.subStudies and
        study = ss.id and
        cite = st.citation and
        ddcodes = (such c in ss.explicitCodes
                   that c.category.name == DesignDecision) and
        effcodes = (such c in ss.explicitCodes
                    that c.category.name == Efficacy) and
        metcodes = (such c in ss.explicitCodes
                    that c.category.name == Method)
   that a==a '
```

APPENDIX 2 CODES USED IN THEMATIC SYNTHESIS

The following tables enumerate all the codes used in the thematic synthesis. The definitions are working ones, used to indicate the limits of the codes for the purposes of coding in this mapping study. In some cases they may be somewhat idiosyncratic. Most codes have been assigned a subcategory as a part of and as a tool in the raw theme formation; they are recorded here for completeness.

Some codes imply other codes; any such implications are indicated at the end of the definition.

TABLE 19 Design-decision codes

Code	Subcategory	Definition
'C	SpecificLanguage	The design decisions under study involve the 'C (tick-C) language.
AJ	SpecificLanguage	The design decisions under study involve the AJ language.
AOP	Paradigm	The study explicitly calls out the aspect-oriented paradigm as one aspect involved in the design decision under study. The aspect-oriented paradigm here refers to the paradigm of programming where cross-cutting concerns are modularized by removing scattered code implementing each such concern into its own module (an aspect).
APL	SpecificLanguage	The design decisions under study involve the APL language.
AWK	SpecificLanguage	The design decisions under study involve the AWK language.
Actors	Multiprocessing	The design decisions under study involve features for the actor model. *Implies* InterprocessMessagePassing.
Ada	SpecificLanguage	The design decisions under study involve the Ada language.
Advice	AOP	The design decisions under study involve advice in aspect-oriented programming. *Implies* AOP.
AggregateOperations	NonVonNeumann	The design decisions under study involve language constructs that express aggregate operations over collections of data.
ArchJava	SpecificLanguage	The design decisions under study involve the ArchJava language.
ArgumentTypeChecking	Typing	The design decisions under study involve static checking of subroutine argument types. *Implies* StaticTyping.
AspectC++	SpecificLanguage	The design decisions under study involve the AspectC++ language.
AspectJ	SpecificLanguage	The design decisions under study involve the AspectJ language.
AssignmentSyntax	Syntax	The design decisions under study involve the syntax for expressing variable assignment (in many languages, either ':=' or '=').
BooleanQueries	Syntax	The design decisions under study involve features for expressing Boolean queries.
Bristlecone	SpecificLanguage	The design decisions under study involve the Bristlecone language.
C	SpecificLanguage	The design decisions under study involve the C language.
C#	SpecificLanguage	The design decisions under study involve the C++ language.
C++	SpecificLanguage	The design decisions under study involve the C++ language.
C++/CORBA	SpecificLanguage	The design decisions under study involve the C++ language with CORBA.
C++/UDP	SpecificLanguage	The design decisions under study involve the C++ language with the UDP and ICI protocol libraries.
C+MPI	SpecificLanguage	The design decisions under study involve the C+MPI language.
CCured	SpecificLanguage	The design decisions under study involve the CCured language.
COBOL	SpecificLanguage	The design decisions under study involve the COBOL language.
CaesarJ	SpecificLanguage	The design decisions under study involve the CaesarJ language.
CallSyntax	Syntax	The design decisions under study involve the syntax for expressing subroutine (including object method) calls (in many languages, a parenthesized list of arguments after the subroutine name).
ClassInheritance	OOFeature	The design decisions under study involve class inheritance as that term is used in the object-oriented context. The boundary between this code and OOPolymorphism is fuzzy, although this term typically focuses on the use of inheritance as a structuring and reuse tool.
Comments	Syntax	The design decisions under study involve language features for code commenting.
CommonLisp	SpecificLanguage	The design decisions under study involve the Common Lisp language.
CompensationStacks	Exceptions	The design decisions under study involve compensation stacks.
Concurrency	Multiprocessing	The design decisions under study involve features for concurrency. Note that concurrency is distinct from parallelism: the former is a program structuring tool (primarily to make the program clearer), the latter is a tool for implementing algorithms (primarily to improve performance).
ConditionalCompilation	ProgrammingTechniqueSupport	The design decisions under study involve language features for conditional compilation.
Conditionals	Syntax	The design decisions under study involve the conditional statement or some other language features for expressing data-directed choice. Loops are not included in this code.
Cyclone	SpecificLanguage	The design decisions under study involve the Ruby language.
DRuby	SpecificLanguage	The design decisions under study involve the DRuby (Diamondback Ruby) language.
DataCentricSynchronization	Multiprocessing	The design decisions under study involve features for designating a set of memory locations as an atomic set and program units as atomic with respect to one or more such atomic sets. *Implies* SharedMemoryCommunication.
DataQueries	NonVonNeumann	The design decisions under study involve language constructs for expressing queries directed to collections of data.

(continues)

TABLE 19 (continues)

Code	Subcategory	Definition
DeclarativeParadigm	Paradigm	The study explicitly calls out the declarative paradigm as one aspect involved in the design decision under study. The declarative paradigm here refers collectively to the paradigms of programming in which the program is predominantly written by expressing declarative properties required of the program. This includes logic programming and functional programming, but this code should only be used when the study uses the term "declarative" explicitly to describe the paradigm.
DeterministicParallelJava	SpecificLanguage	The design decisions under study involve the Deterministic Parallel Java language
DeterministicParallelism	Multiprocessing	The design decisions under study involve features for deterministic parallelism. *Implies* Parallelism.
DynamicFaultDiagnosis	SafetyFeature	The design decisions under study involve the diagnosis of dynamic faults (such as array bounds violations).
DynamicTyping	Typing	The design decisions under study involve dynamic typing (that is, language features that make distinctions between types and enforce those distinctions dynamically by diagnosing those type errors in a program that are about to occur in a particular execution, and preventing the type error from occurring usually by diverting the control flow or by aborting the program).
EJFlow	SpecificLanguage	The design decisions under study involve the EJFlow language.
EJP	AOP	The design decisions under study involve explicit join points in aspect-oriented programming. *Implies* AOP.
ESJ	SpecificLanguage	The design decisions under study involve the Eventful Session Java language.
ET	SpecificLanguage	The design decisions under study involve the ET (Energy Types) language.
Eiffel	SpecificLanguage	The design decisions under study involve the Eiffel language.
EnergyAwareness	ApplicationArea	The design decisions under study involve features for controlling energy usage.
EqualitySyntax	Syntax	The design decisions under study involve the syntax for expressing equality testing (in many languages, either '==' or '=').
Erlang	SpecificLanguage	The design decisions under study involve the Erlang language.
EventDrivenProgramming	ProgrammingTechniqueSupport	The design decisions under study involve features for event-driven programming.
ExceptionHandling	Exceptions	The design decisions under study involve features, such as try..catch, for handling exceptional situations outside the normal control flow.
F#	SpecificLanguage	The design decisions under study involve the F# language.
FP	Paradigm	The study explicitly calls out the functional paradigm as one aspect involved in the design decision under study. The functional paradigm here refers to the paradigm of programming where programs are predominantly composed from existing functions by the aid of functionals (higher-order functions); often the functions are pure (that is, lack side effects).
FamilySharing	OOFeature	The design decisions under study involve features for family sharing of interface types. *Implies* StaticTyping.
FeatureDesign	DecisionType	A study in which multiple mutually exclusive design choices for a particular language feature are compared, or otherwise the design of a particular feature is at issue.
FeaturePresence	DecisionType	A study in which the design decision is the presence or absence of a particular set of features.
Fixity	Syntax	The design decisions under study involve the fixity of operators (not just in evaluable expressions but possibly also in commands and statements). Fixities include prefix (operator before operands), infix (operator between operands), and postfix (operator after operands).
Focus	SpecificLanguage	The design decisions under study involve the Focus language.
Forth	SpecificLanguage	The design decisions under study involve the Forth language.
Fortran	SpecificLanguage	The design decisions under study involve the Fortran language.
FourthGeneration	LanguageGeneration	A study that explicitly calls out the fourth programming language generation as one of the objects of the study.
GOTO	Syntax	The design decisions under study involve the unconditional, unresticted jump statement (GOTO). This does not include studies in which a GOTO statement's use is restricted (such as studies of IF...GOTO, which should be coded as Conditionals.)
GRAIL	SpecificLanguage	The design decisions under study involve the GRAIL language.
GarbageCollection	MemoryManagement	The design decisions under study involve automatic memory management via garbage collection.
GdH	SpecificLanguage	The design decisions under study involve the Glasgow Distributed Haskell (GdH) language.
GenerationComparison	DecisionType	A study that explicitly calls out programming language generations as the objects of the study.
Griffin	SpecificLanguage	The design decisions under study involve the Griffin language.
Groovy	SpecificLanguage	The design decisions under study involve the Groovy language.
HJ	SpecificLanguage	The design decisions under study involve the Habanero Java language.
HJp	SpecificLanguage	The design decisions under study involve the Habanero Java With Permissions language.
Haskell	SpecificLanguage	The design decisions under study involve the Haskell language.
Hypertalk	SpecificLanguage	The design decisions under study involve the Hypertalk language.
ITD	AOP	The design decisions under study involve intertype declarations in aspect-oriented programming. *Implies* AOP.
IndirectRelevance		The study has no explicit or implicit programming language design decision under study, but its results are transferable.
InterprocessMessagePassing	Multiprocessing	The design decisions under study involve features for interprocess communication via message-passing
Iterators	ProgrammingTechniqueSupport	The design decisions under study involve iterators (a set of language features for expressing traversal algorithms for specific data structures, to be used in directing a foreach-style loop).
J&	SpecificLanguage	The design decisions under study involve the J& language.
J&h	SpecificLanguage	The design decisions under study involve the J&h language.
JPred	SpecificLanguage	The design decisions under study involve the JPred language.

(continues)

TABLE 19 (continues)

Code	Subcategory	Definition
JUMP	SpecificLanguage	The design decisions under study involve Sime et al's JUMP microlanguage.
JUMP-M	SpecificLanguage	The design decisions under study involve Vessey and Weber's JUMP-M microlanguage, which is based on Sime et al's JUMP.
Java	SpecificLanguage	The design decisions under study involve the Java language.
JoinCalculusFeatures	Multiprocessing	The design decisions under study involve features derived from the join calculus (e.g. chords in Polyphonic C#)
KAILSelector	Syntax	The design decisions under study involve the KAIL selector.
Klerer–May	SpecificLanguage	The design decisions under study involve the Klerer–May language.
LOGO	SpecificLanguage	The design decisions under study involve the LOGO language.
Lambda	FPFeature	The design decisions under study involve lambda expressions (that is expressions evaluating to function values, where the function's parameters and body are included in the expression itself).
LanguageComparison	DecisionType	A study in which two full languages are compared. This code should not be used if the languages in question were designed as research vehicles and have no use in actual programming (the paradigmatic cases being Sime et al's languages JUMP, NEST, etc).
Loops	Syntax	The design decisions under study involve looping statements such as while statements.
ML	SpecificLanguage	The design decisions under study involve the ML language family (including SML and OCaml but excluding Haskell).
MPI	SpecificLanguage	The design decisions under study involve the MPI library.
ManualDelete	MemoryManagement	The design decisions under study involve manual memory management, specifically manual deletion of unused memory objects.
MemoryLocking	Multiprocessing	The design decisions under study involve features for in-memory locks (including semaphores, mutexes and MVars). *Implies* SharedMemoryCommunication.
Microprogramming	ApplicationArea	The design decisions under study involve microprogramming.
Modula-2	SpecificLanguage	The design decisions under study involve the Modula-2 language.
MorphJ	SpecificLanguage	The design decisions under study involve the MorphJ language.
Multithreading	Multiprocessing	The design decisions under study involve features for managing multiple threads of control (that is, multiple simultaneous but independent control flows with a shared memory).
NEST	SpecificLanguage	The design decisions under study involve Sime et al's NEST microlanguage.
NEST-BE	SpecificLanguage	The design decisions under study involve Sime et al's NEST-BE microlanguage.
NEST-INE	SpecificLanguage	The design decisions under study involve Sime et al's NEST-INE microlanguage.
NLC	SpecificLanguage	The design decisions under study involve the NLC system for natural language programming.
NameOverloading	ProgrammingTechniqueSupport	The design decisions under study involve features for the programmer to introduce new meanings to existing names with multiple meanings being available at the same time, disambiguated by the context of each use. Also called ad hoc polymorphism.
NestedIntersection	OOFeature	The design decisions under study involve features for nested intersection of interface types. *Implies* StaticTyping.
NestedParallelism	Multiprocessing	The designg decisions under study involve the ability to use parallelism within a single thread to implement parallel algorithms. *Implies* Parallelism, STM, SharedMemoryCommunication.
NestedSubroutines	ProgrammingTechniqueSupport	The design decisions under study involve the ability to define subroutines within subroutines (like in Pascal but unlike in C).
NeverNullReferences	SafetyFeature	The design decisions under study involve features for declaring and enforcing the non-nullity of reference variables.
NondeterministicParallelism	Multiprocessing	The design decisions under study involve features for nondeterministic parallelism. *Implies* Parallelism.
OOP	Paradigm	The study explicitly calls out the object-oriented paradigm as one aspect involved in the design decision under study. The object-oriented paradigm here refers to the paradigm of programming where the program is decomposed into objects possessing identity, behaviour and state, and potentially related by some sort of incremental modification device such as inheritance, and they are composed to form the program by having the objects invoke each others' methods. Support for classes is not a requirement, but it is common.
OOPolymorphism	OOFeature	The design decisions under study involve polymorphism as that term is used in the object-oriented context. This typically includes subtype polymorphism via class inheritance and dynamic binding of class methods and typically focuses on the ability to treat objects of inheritance-related classes as interchangeable, resulting in differences in run-time behaviour based on the run-time classes of the involved objects. Note that the word 'polymorphism' has a different meaning in the type-theoretic and functional-programming context.
OT/J	SpecificLanguage	The design decisions under study involve the ObjectTeams/Java language.
ObjectConstructors	OOFeature	The design decisions under study involve features for the programmer to customize object construction.
ObjectImmutability	OOFeature	The design decisions under study involve features in which an object, once created, cannot change its attributes.
ObjectPascal	SpecificLanguage	The design decisions under study involve the Object Pascal language.
Objective-C	SpecificLanguage	The design decisions under study involve the Objective-C language.
PL/C	SpecificLanguage	The design decisions under study involve the PL/C language.
PLTScheme	SpecificLanguage	The design decisions under study involve the PLT Scheme language.
ParadigmComparison	DecisionType	A study in which two programming or language paradigms are compared, by intent if not in fact (usually operationalized into LanguageComparison)
Parallelism	Multiprocessing	The design decisions under study involve features for parallelism. Note that parallelism is distinct from concurrency: the former is a tool for implementing algorithms (primarily to improve performance), the latter is a program structuring tool (primarily to make the program clearer).
Pascal	SpecificLanguage	The design decisions under study involve the Pascal language.

(continues)

TABLE 19 (continues)

Code	Subcategory	Definition
Perl	SpecificLanguage	The design decisions under study involve the Perl language.
PermissionTypes	Typing	The design decisions under study involve static typing features for access permission control. *Implies* StaticTyping.
Pointcuts	AOP	The design decisions under study involve point-cuts in aspect-oriented programming. *Implies* AOP.
PredicateDispatch	ProgrammingTechniqueSupport	The design decisions under study involve predicate dispatch (choosing which definition of a named function to apply based on predicates on parameters attached to each definition).
ProceduralParadigm	Paradigm	The study explicitly calls out the procedural paradigm as one aspect involved in the design decision under study. The procedural paradigm here refers to the paradigm of programming where the program is functionally decomposed into subroutines that are invoked by name and that may take parameters and may also return a value and in which side-effects are possible and accepted.
ProgramIndentation	Syntax	The design decisions under study involve indentation of programs to make program structure visible.
Prolog	SpecificLanguage	The design decisions under study involve the Prolog language.
Proteus	SpecificLanguage	The design decisions under study involve the Proteus language.
Python	SpecificLanguage	The design decisions under study involve the Python language.
Quorum	SpecificLanguage	The design decisions under study involve the Quorum language.
RTSJ	SpecificLanguage	The design decisions under study involve the RTSJ (Real-Time Specification for Java) language.
Randomo	SpecificLanguage	The design decisions under study involve the Randomo language.
Rapide	SpecificLanguage	The design decisions under study involve the Rapide language.
RelationalLisp	SpecificLanguage	The design decisions under study involve the RelationalLisp language.
Rexx	SpecificLanguage	The design decisions under study involve the Rexx language.
Ruby	SpecificLanguage	The design decisions under study involve the Ruby language.
RuntimeCodeGeneration	ProgrammingTechniqueSupport	The design decisions under study involve features for runtime code generation.
SIMONE	SpecificLanguage	The design decisions under study involve the SIMONE language.
SJ	SpecificLanguage	The design decisions under study involve the Session Java language.
SML	SpecificLanguage	The design decisions under study involve the SML language. *Implies* ML.
STARS	SpecificLanguage	The design decisions under study involve the STARS (Scoped Types and Aspects for Real-Time Systems) language.
STM	Multiprocessing	The design decisions under study involve software transactional memory features. *Implies* SharedMemoryCommunication.
STRUM	SpecificLanguage	The design decisions under study involve the STRUM language.
Scala	SpecificLanguage	The design decisions under study involve the Scala language.
ScopeDelimiters	Syntax	The design decisions under study involve scope-delimiting constructs (such as begin..end).
ScriptingParadigm	Paradigm	The study explicitly calls out scripting languages as one aspect involved in the design decision under study.
SecurityIssuePrevention	SafetyFeature	The design decisions under study involve features intended to prevent security holes or other security issues.
SelfAdjustingComputation	ProgrammingTechniqueSupport	The design decisions under study involve self-adjusting computation features – ways to allow the programmer to write a batch algorithm while having it be translated to incremental computation, that is, computation that is able to modify its output given changes to input with less effort than recomputing from scratch.
SharedMemoryCommunication	Multiprocessing	The design decisions under study involve features for interprocess communication via a shared memory with some kind of synchronization discipline.
SideEffectingExpressions	SafetyFeature	The design decisions under study involve expressions which have side effects.
StatementSequencingSyntax	Syntax	The design decisions under study involve the syntax for expressing statement sequencing (in many languages, the semicolon, either as a terminator as in C or as a separator as in Pascal).
StaticTyping	Typing	The design decisions under study involve static typing (that is, language features that make distinctions between types and enforce those distinctions statically by diagnosing all type errors in a program before any execution starts, usually at compile time).
StreamIt	SpecificLanguage	The design decisions under study involve the StreamIt language.
StreamProgramming	Paradigm	The design decisions under study involve stream programming.
StringConcatenationSyntax	Syntax	The design decisions under study involve the syntax for expressing string concatenation.
StringLiteralSyntax	Syntax	The design decisions under study involve the syntax for expressing literal strings (in many languages, enclosure in quotes).
StructuralSubtyping	Typing	The design decisions under study involve static typing features for structural subtyping. *Implies* StaticTyping.
StructuredProgramming	Paradigm	The study explicitly calls out the structured programming paradigm as one aspect involved in the design decision under study. The structured programming paradigm here refers to the paradigm of programming where programs are structured using block-structuring constructs such as while loops, if-then-else-constructs and parameter-passing subroutines. Normally, this code is used without the ParadigmComparison code, as there is usually no defined paradigm as control.
SystemProgrammingParadigm	Paradigm	The study involves system programming languages, which are defined as the complement of scripting languages. This code should only be used in conjunction with the ScriptingParadigm code.
TOPPS	SpecificLanguage	The design decisions under study involve the TOPPS language.
TRANSLANG	SpecificLanguage	The design decisions under study involve the TRANSLANG language.
TaskSpecificConstructs	NonVonNeumann	The design decisions under study involve language constructs specifically designed to embody task-specific algorithms such as sorting or searching.
Tcl	SpecificLanguage	The design decisions under study involve the Tcl language.
ThirdGeneration	LanguageGeneration	A study that explicitly calls out the third programming language generation as one of the objects of the study.

(continues)

TABLE 19 (continues)

Code	Subcategory	Definition
TupleType	Typing	The design decisions under study involve tuple types (that is, finite heterogeneous sequences in which the length of the sequence and the types of the individual sequence members are fixed by the tuple type). *Implies* StaticTyping.
TypeCasting	Typing	The design decisions under study involve type casting (that is, explicit type conversions, usually requiring either dynamic checking or a change of representation at runtime). *Implies* StaticTyping.
TypeInference	Typing	The design decisions under study involve type inference (that is, static typing where the type system infers the vast majority of types that one would normally expect to declare explicitly). *Implies* StaticTyping.
TypeQualifiers	Typing	The design decisions under study involve type qualifier features.
TypedScheme	SpecificLanguage	The design decisions under study involve the Typed Scheme language.
TypelessLanguage	Typing	The design decisions under study involve a typeless language (that is, a language that makes no type distinctions and in which therefore type errors are a meaningless concept). Do not confuse this with' DynamicTyping.
UPC	SpecificLanguage	The design decisions under study involve the UPC language.
Umple	SpecificLanguage	The design decisions under study involve the Umple language.
VBA	SpecificLanguage	The design decisions under study involve the Visual Basic for Applications language.
ValueNotIgnorable	FPFeature	The design decisions under study include the overall principle that the value of an expression cannot be ignored (like one does in, for example, C's expression statements).
X10	SpecificLanguage	The design decisions under study involve the X10 language.
XAML	SpecificLanguage	The design decisions under study involve the XAML language.
XMTC	SpecificLanguage	The design decisions under study involve the XMTC language.
occam	SpecificLanguage	The design decisions under study involve the occam language.

TABLE 20 Efficacy codes

Code	Subcategory	Definition
AnnotationOverhead	Effort	In a FeaturePresence study, a program written without the feature can often be annotated to exploit the presence of the feature. This code indicates that the study measures the annotation overhead (typically as the percentage of added lines of code) of the feature.
ClozeTestPerformance	ProgramComprehension	Efficacy is measured in the study by having participants fill holes left in an otherwise complete program.
DebuggingEffort	Effort	Debugging (locating and fixing errors) effort is measured in the study.
DesignStability	QualityAttributes	This study measures design stability over maintenance periods.
ErrorProneness	Correctness	This study measures error proneness of the design decisions at stake. Includes studies where correctness of participant-written programs is measured.
FeaturePrevalence	ActualUsage	The study examines how much a particular feature is used in the existing code base. This may indicate a post-hoc EXCLUDE candidate (since this is not really an efficacy measure).
Learnability	Learnability	It is measured in the study, how easy or hard the design choices involved are to learn.
LinesOfCodeComparison	Effort	Efficacy is measured, in part or in full, by comparing the lines-of-code sizes of different programs representing the different design choices involved.
MaintenanceEffort	Effort	Maintenance effort (including reengineering or refactoring) is measured in the study.
Modifiability	QualityAttributes	Program modifiability is measured in the study.
Modularity	QualityAttributes	This study measures program modularity.
PerceivedComplexity	Subjective	This study measures language complexity as perceived by the participants.
PerceivedIntuitivity	Subjective	This study measures the intuitivity of the design decision at issue as perceived by the participants.
PerceivedValue	Subjective	The study examines perceptions of the value of the design decisions at issue.
ProgramComprehension	ProgramComprehension	Program comprehension is measured in the study, usually by proxy via scores attained by experiment participants in a comprehension test.
ProgramQuality	QualityAttributes	This study measures program quality. This usually means the use of QualityMetrics.
ProgramTranslationEffort	Effort	The effort to manually translate programs from one design choice to another is measured in the study.
ProgrammingEffort	Effort	Programming effort is measured in the study, usually by proxy via either wall-clock time or lines of code.
RetrofittingOpportunity	ActualUsage	The study examines how well the usage of the language in the existing code base for a language would allow retrofitting the design decision to the language.
Reusability	QualityAttributes	Code reusability is measured in the study.
RuntimePerformance	Performance	Runtime performance (typically runnning time, memory usage, or scalability) is measured in the study.
SecurityVulnerabilityProneness	Correctness	This study measures the proneness for security vulnerabilities of the design decisions at stake. *Implies* ErrorProneness.
Testability	QualityAttributes	Program testability is measured in the study.
UnspecifiedEfficacy		Efficacy is not specified in the report. This code will generally only be used for coding secondary studies' recounting of primary studies, which sometimes neglect to discuss the primary study in detail. If this code occurs in a primary study, the study ought to be given a post-hoc EXCLUDE.

TABLE 21 Method codes for primary studies

Code	Subcategory	Definition
AdvancedProgrammingStudents	ParticipantClass	The participants in the study are students of programming taking advanced programming courses, in the late stages of an undergraduate degree in computer science (CS) or software engineering (SE), or pusuing a graduate degree in CS or SE. Information Systems (IS) students commonly do not get enough programming training to qualify even in the postgraduate level. *Implies* HumanParticipants, Programmers, ProgrammingStudents.
ArtifactEncoding	DataAnalysis	A particular program correctness analysis method, similar to how exams would be graded.
BeginningProgrammingStudents	ParticipantClass	The participants in the study are students of programming who are taking or have completed basic programming courses but no advanced courses in programming and no (other) courses in which programming skill is exercised. *Implies* HumanParticipants, Programmers, ProgrammingStudents.
BenchmarkPrograms	PrimaryMethod	A number of programs are adopted from the literature or folklore to serve as specimens demonstrating the efficacy of the design decisions under study. The programs may be modified or even completely rewritten for the study, but the point is to demonstrate the language's capabilities, not to measure the human-induced contingencies involved in the modification or rewriting activity. Studies marked with this code are candidates for post-hoc EXCLUDE decisions for lack of empiricity (such studies have a strong analytical feel). Contrast to CorpusAnalysis, in which a large number of existing applications are selected (usually with a rational and explicit set of selection criteria) and analyzed as they are, without any rewriting or modification Contrast to ProgramRewriting, in which the act of rewriting programs is the main interest. *Implies* ResearcherParticipates.
BetweenSubjects	ExperimentDesign	In a controlled experiment, results are obtained by comparing the performance of subjects to one another (usually by comparing the aggregated performance of the various groups). *Implies* ControlledExperiment, Experiment.
BugHistory	DataSource	Study that examines the recorded history of bugs in a particular program, usually via a bug tracking system's historical records.
CaseStudy	PrimaryMethod	Study that examines an entity (program, organization etc.) that exists independently of the study without experimental manipulation and within the entity's own context. Note that many studies label themselves as case studies even though they do not fit this definition, and thus should not be coded CaseStudy.
CodeHistory	DataSource	Study that examines the recorded history of a particular program, usually via version control logs or via a sequence of public releases.
ControlledExperiment	ExperimentDesign	An experiment in which experimental subjects or specimens (called participants, if they are persons) allocated into different groups based on the values of independent variables imposed to or inherent in the experimental subjects as well as the possible sequences in which the independent variables are manipulated within a single group, in such a way that all relevant distinctions of the values of the independent variables and manipulation sequences are accounted for in the allocation. Results are typically obtained by statistical analysis, treating the experimental units as having been randomly sampled from some population (whether this is actually true or not varies). *Implies* Experiment.
CorpusAnalysis	PrimaryMethod	Study is centered around analyzing a (usually large) set of programs written for other purposes than the study in question Contrast to BenchmarkPrograms, in which a number of existing programs are modified or even rewritten to suit the analysis. Contrast to ProgramPairAnalysis, in which related pairs of programs are analyzed. Contrast to ProgramRewrite in which existing programs are rewritten to produce data.
EndUserProgrammers	ParticipantClass	The participants in the study are people who are neither students of programming nor professional programmers nor serious programming hobbyists but who do program to accomplish their (non-programming-related) tasks, mostly only for a limited audience. *Implies* HumanParticipants, Programmers.
Experiment	PrimaryMethod	Study in which the relationship of one or more (dependent) variables to one or more (independent) variables is investigated in a setting controlled by the researchers, by manipulating the independent variables and observing the dependent variables and attempting to keep all other (confounding) variables constant.
GroundedTheory	PrimaryMethod	A particular research method. In this study, we take the researchers' word for it: code GroundedTheory iff the researchers claim to have used it in a relevant way.
HistoricalControl	DataSource	A study, usually a controlled experiment, in which the control group's data are gathered from past measurements not connected with the present study.
HumanParticipants	DataSource	Study in which the behaviour of humans, usually specifically recruited by the researchers for the study, is observed, with or without influence from the researchers.
Interviews	DataSource	Study in which one or more human participants are interviewed. *Implies* HumanParticipants.
LanguageShootout	ExperimentDesign	The same programming task is handed out to different programmers or programming teams, each implementing it in a particular language. *Implies* BetweenSubjects, ControlledExperiment, Experiment, HumanParticipants.
MetricsCollectionAnalysis	DataAnalysis	Study is centered around analyzing a (usually large) set of program or project metrics collected for some other purpose than this particular study.
NonProgrammers	ParticipantClass	The participants in the study are people who have no programming background and are not studying programming at the time of the study. *Implies* HumanParticipants.
NonrandomizedControlledExperiment	ExperimentDesign	A controlled experiment in which the experimental subjects are allocated to groups using a non-random process. *Implies* ControlledExperiment, Experiment.
OpenCoding	DataAnalysis	Codes (labels, tags) for particular meanings that emerge from the data are assigned by the researcher. Similar to what is being done here.

(continues)

TABLE 21 (continues)

Code	Subcategory	Definition
OtherExperiment	ExperimentDesign	An experiment that is not a controlled experiment. *Implies* Experiment.
ProfessionalProgrammers	ParticipantClass	The participants in the study are professional programmers. This includes professional testers and teachers of programming (preparing students for professional programming). It also includes hobbyist programmers with extensive experience comparable to those of professionals. *Implies* HumanParticipants, Programmers.
ProgramPairAnalysis	PrimaryMethod	Study is centered around analyzing one or more (usually no more than a handful of) pairs of programs written for other purposes than the study in question, such that the programs in each pair are related in some relevant manner (for example, they implement the same spec but are written in different languages). It is distinguished from CorpusAnalysis by the use of related pairs, and by the usually small number of pairs. It is distinguished from ProgramRewrite by the fact that the programs are preexisting.
ProgramRewrite	PrimaryMethod	An existing program (or a handful of such) is rewritten from an established language to a new language.
ProgrammerObservation	DataSource	A study in which the behaviour of programmers during actual work is observed. *Implies* HumanParticipants, Programmers.
Programmers	ParticipantClass	The participants have some training or experience is programming, however minimal or extensive. *Implies* HumanParticipants.
ProgrammingStudents	ParticipantClass	The participants in the study are students of programming pursuing an undergraduate or graduate degree in Computer Science or Software Engineering, or are otherwise taking training designed to prepare its students for professional programming. *Implies* HumanParticipants, Programmers.
QualityMetrics	Metrics	Published code quality metrics are used to measure efficacy.
RandomizedControlledExperiment	ExperimentDesign	A controlled experiment in which the experimental subjects are allocated to groups using a random process. *Implies* ControlledExperiment, Experiment.
ReAnalysis	PrimaryMethod	Data collected for a previous study is analyzed anew. Any other method codes accompanying this code relate to the original study.
RegressionTesting	DataSource	Study that uses automated regression testing to identify problems in a particular program.
ResearcherParticipates	DataSource	One of the researchers is a significant participant in the activity under study (for example, writing a program).
SimulatedMaintenance	DataSource	Maintenance tasks are given to the participants.
SingleSubjectExperiment	ExperimentDesign	An experiment where a single subject functions as the only experimental unit. Obviously, none of the idependent variables can be inherent in the unique subject. Usually, such experiments are designed so that each relevant combination of independent variables is applied to the subject in sequence, sometimes more than once, and the dependent variables are measured for each such application. Such experiments cannot be coded as ControlledExperiments, as it is not possible to create groups that try out the various orderings of the applications. *Implies* Experiment.
SoftwareScience	Metrics	Study explicitly invokes the "software science" body of work, and uses its metrics in a significant way.
StaticAnalysis	DataSource	The programs under study are fed to a static analyzer embodying the language design choices at issue.
Survey	PrimaryMethod	Study in which a group of human participants are asked to fill a questionnaire. Results are typically obtained by statistical analysis, treating the participants as having been randomly sampled from some population (whether this is actually true or not varies), but qualitative analyses are sometimes employed. *Implies* HumanParticipants.
UnspecifiedMethod		Method is not specified in the report, or it is so badly described that it cannot be meaningfully coded. This code will generally only be used for coding secondary studies' recounting of primary studies, which sometimes neglect to discuss the primary study methodology. If this code occurs in a primary study, the study ought to be given a post-hoc EXCLUDE.
WithinSubjects	ExperimentDesign	In a controlled experiment, results are obtained by measuring the performance of each subject more than once, with independent variables being manipulated in between, and comparing these repeated measures within each subject. *Implies* ControlledExperiment, Experiment.

TABLE 22 Method codes for secondary studies

Code	Subcategory	Definition
NarrativeReview		A literature review in which the review method is not specified (except to the extent indicated by other SecondaryMethod codes).
SearchDatabasesSpecified		The literature databases in which keyword searches were conducted are reported.
SearchTermsSpecified		The terms used to search literature in keyword searches are reported.

APPENDIX 3 CODE ASSIGNMENTS FOR INCLUDED STUDIES

TABLE 23 Primary studies and their assigned codes

Study	Citation	Design decision codes	Efficacy codes	Method codes
S1	Ahmad and Talha 2002	LanguageComparison, Prolog, C++, ParadigmComparison, ProceduralParadigm, DeclarativeParadigm	ProgrammingEffort	CorpusAnalysis, SoftwareScience
S2	Ahsan et al. 2009	LanguageComparison, C, C++, Java	ErrorProneness	CaseStudy, CodeHistory, BugHistory
S3	Aldrich et al. 2002	LanguageComparison, Java, ArchJava	ProgramTranslationEffort, LinesOfCodeComparison	ProgramRewrite, ResearcherParticipates
S4	Andreae et al. 2006	LanguageComparison, STARS, RTSJ	RuntimePerformance	ProgramRewrite, ResearcherParticipates
S6	Badreddin, Forward, et al. 2012; Badreddin and Lethbridge 2012	LanguageComparison, Java, Umple	ProgramComprehension	NonrandomizedControlledExperiment, HumanParticipants
S7	Badreddin and Lethbridge 2012	LanguageComparison, Umple, Java	PerceivedComplexity	Survey, Interviews, GroundedTheory
S8	Badri et al. 2012	LanguageComparison, Java, AspectJ, ParadigmComparison, OOP, AOP	Testability	ProgramPairAnalysis, QualityMetrics
S9	Barnes and Welch 2001	FeaturePresence, occam, InterprocessMessagePassing	RuntimePerformance	BenchmarkPrograms
S10	Bartsch and Harrison 2008	ParadigmComparison, AOP, OOP, LanguageComparison, Java, AspectJ	MaintenanceEffort, ProgramComprehension, Modifiability	RandomizedControlledExperiment, ProfessionalProgrammers, BetweenSubjects
S11a	Benander and Benander 1997	LanguageComparison, Pascal, C	Learnability	Survey, BeginningProgrammingStudents
S11b	Benander and Benander 1997	LanguageComparison, Pascal, C	ErrorProneness	NonrandomizedControlledExperiment, BeginningProgrammingStudents
S12	Benton et al. 2004	FeaturePresence, C#, JoinCalculusFeatures	RuntimePerformance	BenchmarkPrograms
S13	Biermann et al. 1983	LanguageComparison, NLC, PL/C	ErrorProneness	NonrandomizedControlledExperiment, WithinSubjects, BeginningProgrammingStudents
S14	Bocchino et al. 2011	FeaturePresence, DeterministicParallelJava, NondeterministicParallelism	AnnotationOverhead, LinesOfCodeComparison	BenchmarkPrograms
S17	Burckhardt et al. 2011	FeaturePresence, DeterministicParallelism, SelfAdjustingComputation, C#	RuntimePerformance	BenchmarkPrograms
S18	Cacho et al. 2009	LanguageComparison, Java, AspectJ, EJFlow, ExceptionHandling	ProgrammingEffort, ErrorProneness, ProgramComprehension	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S20	Cartwright 1998	FeaturePresence, ClassInheritance	MaintenanceEffort, ProgramComprehension	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S21	Castor, Cacho, et al. 2009	LanguageComparison, AspectJ, Java, ParadigmComparison, AOP, OOP, ExceptionHandling	ProgramQuality, Reusability	ProgramRewrite, QualityMetrics, ResearcherParticipates
S22	Castor, Oliveira, et al. 2011	FeatureDesign, SharedMemoryCommunication, Haskell, STM, MemoryLocking	ErrorProneness, ProgrammingEffort, LinesOfCodeComparison	RandomizedControlledExperiment, AdvancedProgrammingStudents, BetweenSubjects
S23	Cesarini et al. 2008	LanguageComparison, Erlang, Java, C#, Python, Ruby	LinesOfCodeComparison, RuntimePerformance	CorpusAnalysis
S24	Chalin and James 2007	FeaturePresence, NeverNullReferences	RetrofittingOpportunity	ProgramRewrite, ResearcherParticipates
S25	Champeaux et al. 1992	ParadigmComparison, OOP, ProceduralParadigm, LanguageComparison, C++, C	PerceivedValue, PerceivedComplexity	CaseStudy, ResearcherParticipates, ProfessionalProgrammers
S26	Charles et al. 2005	LanguageComparison, X10, Java	MaintenanceEffort	BenchmarkPrograms
S27	Chen and Vecchio 1992	FeatureDesign, Conditionals	ProgramComprehension	OtherExperiment, BeginningProgrammingStudents
S28	Cherry 1986	FeatureDesign, Fixity, IndirectRelevance	ProgrammingEffort, ErrorProneness	RandomizedControlledExperiment, BetweenSubjects, NonProgrammers
S29	Coelho et al. 2008	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java, ExceptionHandling	ErrorProneness	ProgramPairAnalysis
S30	Cohen et al. 2012	LanguageComparison, Java, ET, FeaturePresence, EnergyAwareness, FeatureDesign, StaticTyping	ProgramTranslationEffort, RuntimePerformance	ProgramRewrite, ResearcherParticipates
S31	Condit et al. 2003	LanguageComparison, C, CCured, StaticTyping, SecurityIssuePrevention	RuntimePerformance, ProgramTranslationEffort	ProgramRewrite, ResearcherParticipates
S33	Daly et al. 1995; Daly et al. 1996	FeatureDesign, ClassInheritance	MaintenanceEffort	RandomizedControlledExperiment, ProgrammingStudents, WithinSubjects
S34	Daly, Sazawal, et al. 2009	FeatureDesign, StaticTyping, DynamicTyping, LanguageComparison, Ruby, DRuby	DebuggingEffort	RandomizedControlledExperiment, WithinSubjects, ProfessionalProgrammers, OpenCoding

(continues)

TABLE 23 (continues)

Study	Citation	Design decision codes	Efficacy codes	Method codes
S36	Demsky and Dash 2008	LanguageComparison, Bristlecone, Java	ErrorProneness, LinesOfCodeComparison	BenchmarkPrograms
S37	Dolado et al. 2003	FeaturePresence, SideEffectingExpressions	ProgramComprehension	NonrandomizedControlledExperiment, AdvancedProgrammingStudents, ProfessionalProgrammers, WithinSubjects
S38a	Dolby et al. 2012	LanguageComparison, AJ, Java, FeaturePresence, DataCentricSynchronization	ProgramTranslationEffort, AnnotationOverhead	ProgramRewrite, ResearcherParticipates
S38b	Dolby et al. 2012	LanguageComparison, AJ, Java, FeaturePresence, DataCentricSynchronization	AnnotationOverhead	ProgramRewrite, ResearcherParticipates
S38c	Dolby et al. 2012	LanguageComparison, AJ, Java, FeaturePresence, DataCentricSynchronization	RuntimePerformance	BenchmarkPrograms
S39	Doscher 1990	LanguageComparison, Ada, C	ErrorProneness, LinesOfCodeComparison, ProgramQuality	ProgramPairAnalysis
S41	Dyer et al. 2012	ParadigmComparison, OOP, AOP, FeatureDesign, Pointcuts	ProgramQuality	QualityMetrics, OtherExperiment, SimulatedMaintenance, ResearcherParticipates
S42	Ebcioğlu et al. 2006	LanguageComparison, C+MPI, UPC, X10	ProgrammingEffort	NonrandomizedControlledExperiment, ProgrammingStudents, BetweenSubjects
S43	Embley 1978	FeatureDesign, Conditionals, Loops, KAILSelector	ProgramComprehension	NonrandomizedControlledExperiment, WithinSubjects, BeginningProgrammingStudents
S44	Endrikat and Hanenberg 2011	ParadigmComparison, OOP, AOP, LanguageComparison, Java, AspectJ	MaintenanceEffort	RandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents
S45	Engebretson and Wiedenbeck 2002	FeaturePresence, TaskSpecificConstructs, Hypertalk	MaintenanceEffort	RandomizedControlledExperiment, BetweenSubjects, EndUserProgrammers
S46	Ertl 1999	LanguageComparison, Forth, Prolog, Perl, Python, Modula-2, ML, C	LinesOfCodeComparison, RuntimePerformance	CorpusAnalysis
S47	Ferrari et al. 2010	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java, FeaturePresence, Pointcuts, Advice, ITD	ErrorProneness	ProgramPairAnalysis, CodeHistory, RegressionTesting
S48	Ferrett and Offutt 2002	ParadigmComparison, OOP, ProceduralParadigm, LanguageComparison, Fortran, C, C++, Java	Modularity	CorpusAnalysis
S49	Figueiredo et al. 2008	FeaturePresence, AOP, ConditionalCompilation, LanguageComparison, Java, AspectJ	DesignStability	OtherExperiment, SimulatedMaintenance, AdvancedProgrammingStudents
S50	Flanagan et al. 2008	FeaturePresence, SharedMemoryCommunication, FeatureDesign, StaticTyping	AnnotationOverhead	BenchmarkPrograms, StaticAnalysis
S51	Foster et al. 2006	FeaturePresence, TypeQualifiers, FeatureDesign, StaticTyping	RetrofittingOpportunity	CorpusAnalysis, StaticAnalysis
S54	Fähndrich and Leino 2003	FeaturePresence, NeverNullReferences	AnnotationOverhead	BenchmarkPrograms, StaticAnalysis
S55	Gannon and Horning 1975a,b; Gannon 1976	LanguageComparison, TOPPS	ErrorProneness	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S56	Gannon 1977	FeatureDesign, StaticTyping, TypelessLanguage	ErrorProneness	NonrandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents
S57	Gil and Shragai 2009	FeatureDesign, ObjectConstructors	RetrofittingOpportunity	CorpusAnalysis, StaticAnalysis
S58	Gil and Lenz 2010	FeaturePresence, NameOverloading	FeaturePrevalence	CorpusAnalysis
S59	Gilmore and Green 1984	FeatureDesign, Conditionals	ProgramComprehension	RandomizedControlledExperiment, BetweenSubjects, NonProgrammers
S60a	Green 1977	JUMP, NEST-INE, NEST-BE, FeatureDesign, Conditionals	ProgramComprehension	OtherExperiment, ProfessionalProgrammers
S60b	Green 1977	JUMP, NEST-INE, NEST-BE, FeatureDesign, Conditionals	ProgramComprehension	OtherExperiment, ProfessionalProgrammers
S62	Greenwood et al. 2007	ParadigmComparison, OOP, AOP, LanguageComparison, Java, AspectJ, CaesarJ	DesignStability	CaseStudy, CodeHistory
S63	Halverson 1993	FeatureDesign, Conditionals	ErrorProneness	RandomizedControlledExperiment, WithinSubjects, BeginningProgrammingStudents
S64	Hanenberg, Kleinschmager, and Josupeit-Walter 2009; Kleinschmager 2009	ParadigmComparison, AOP, OOP, LanguageComparison, Java, AspectJ	MaintenanceEffort	SimulatedMaintenance, RandomizedControlledExperiment, AdvancedProgrammingStudents, WithinSubjects
S65	Hanenberg 2009, 2010a,b	FeatureDesign, StaticTyping, DynamicTyping	ProgrammingEffort, ErrorProneness	NonrandomizedControlledExperiment, BetweenSubjects, ProgrammingStudents
S66	Harel and McLean 1985	GenerationComparison, ThirdGeneration, FourthGeneration, LanguageComparison, COBOL, Focus	ProgrammingEffort	ProfessionalProgrammers, NonrandomizedControlledExperiment, BetweenSubjects
S67	Harrison, Smaraweera, et al. 1996	ParadigmComparison, FP, OOP, LanguageComparison, SML, C++	ErrorProneness, PerceivedComplexity, Reusability, DebuggingEffort	SingleSubjectExperiment, HumanParticipants

(continues)

TABLE 23 (continues)

Study	Citation	Design decision codes	Efficacy codes	Method codes
S68	Harrison, Counsell, et al. 2000	FeaturePresence, ClassInheritance, C++	Modifiability, ProgramComprehension	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S69	Henry and Humphrey 1988; Henry and Humphrey 1990; Henry and Humphrey 1993	ParadigmComparison, ProceduralParadigm, OOP, LanguageComparison, C, Objective-C	MaintenanceEffort	RandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents
S70	Hertz and Berger 2005	FeatureDesign, GarbageCollection, ManualDelete	RuntimePerformance	BenchmarkPrograms
S71	Hicks et al. 2004	FeatureDesign, GarbageCollection, ManualDelete, Cyclone	RuntimePerformance	BenchmarkPrograms
S72	Hitz and Hudec 1995	LanguageComparison, Modula-2, C++	ErrorProneness	NonrandomizedControlledExperiment, HistoricalControl, BeginningProgrammingStudents, BetweenSubjects
S74	Hochstein and Basili 2006; Hochstein, Basili, et al. 2008	FeatureDesign, Parallelism, InterprocessMessagePassing, SharedMemoryCommunication, LanguageComparison, XMTC, MPI, C++, Fortran	ErrorProneness, ProgrammingEffort	NonrandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S75	Hoffman and Eugster 2008	FeaturePresence, EJP, AspectJ	ProgramQuality, Reusability	QualityMetrics, ProgramRewrite, ResearcherParticipates
S76	Hu et al. 2010	FeaturePresence, EventDrivenProgramming, Multithreading, LanguageComparison, ESJ, SJ	RuntimePerformance	BenchmarkPrograms
S77	Huang and Smaragdakis 2011	LanguageComparison, MorphJ, Java	LinesOfCodeComparison	BenchmarkPrograms
S78	Hudak and Jones 1994	LanguageComparison, Haskell, Ada, C++, AWK, Rapide, Griffin, Proteus, RelationalLisp	LinesOfCodeComparison, ProgrammingEffort	ProfessionalProgrammers, LanguageShootout
S79	Iselin 1988	FeatureDesign, Conditionals, Loops, COBOL	ProgramComprehension	RandomizedControlledExperiment, AdvancedProgrammingStudents, ProfessionalProgrammers, BetweenSubjects
S80	Jim et al. 2002	LanguageComparison, C, Cyclone	LinesOfCodeComparison, ProgramTranslationEffort	ProgramRewrite, ResearcherParticipates
S82	Kesler et al. 1984	FeaturePresence, ProgramIndentation, Pascal	ProgramComprehension	RandomizedControlledExperiment, AdvancedProgrammingStudents, BetweenSubjects
S83	Kleinschmager et al. 2012; Kleinschmager 2012	FeatureDesign, StaticTyping, DynamicTyping, LanguageComparison, Java, Groovy	ProgrammingEffort	RandomizedControlledExperiment, ProgrammingStudents, WithinSubjects
S84	Klerer 1984	LanguageComparison, Klerer–May, Fortran	ProgrammingEffort	RandomizedControlledExperiment, BeginningProgrammingStudents, WithinSubjects
S85	Kosar et al. 2010	LanguageComparison, XAML, C#	ProgramComprehension	UnspecifiedMethod
S86	Kulesza et al. 2006	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java	ProgramQuality	QualityMetrics, ProgramPairAnalysis, SimulatedMaintenance
S88	Leblanc and Fischer 1982	FeaturePresence, DynamicFaultDiagnosis	ErrorProneness	CaseStudy, ProgrammerObservation, ProgrammingStudents
S89	Lee et al. 2003	FeaturePresence, StructuralSubtyping, FeatureDesign, StaticTyping	FeaturePrevalence	CaseStudy, CodeHistory
S90	Lewis et al. 1991, 1992	ParadigmComparison, OOP, ProceduralParadigm, LanguageComparison, C++, Pascal	ProgrammingEffort, Reusability	RandomizedControlledExperiment, AdvancedProgrammingStudents, BetweenSubjects
S91	Lima et al. 2011	LanguageComparison, OT/J, Java	Modularity	CorpusAnalysis
S92	Liu et al. 2006	FeatureDesign, Iterators	LinesOfCodeComparison, ProgramTranslationEffort	ProgramRewrite, ResearcherParticipates
S93	Lucas and Kaplan 1976	FeaturePresence, GOTO	ProgrammingEffort, MaintenanceEffort	RandomizedControlledExperiment, SimulatedMaintenance, BeginningProgrammingStudents, BetweenSubjects
S94	Luff 2009	FeatureDesign, Concurrency, STM, MemoryLocking, Actors	ProgrammingEffort, LinesOfCodeComparison, PerceivedComplexity	NonrandomizedControlledExperiment, WithinSubjects, ProgrammingStudents
S95	Madeyski and Szala 2007	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java	Modularity, LinesOfCodeComparison, ProgrammingEffort	LanguageShootout, AdvancedProgrammingStudents
S96	Malayeri and Aldrich 2009	FeaturePresence, StructuralSubtyping, FeatureDesign, StaticTyping	RetrofittingOpportunity	CorpusAnalysis
S97	Mayer et al. 2012b; Mayer et al. 2012a	FeatureDesign, StaticTyping, DynamicTyping, LanguageComparison, Java, Groovy	ProgrammingEffort	RandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents
S98	McCaffrey and Bonar 2010	FeaturePresence, TypeInference, TupleType, ObjectImmutability, Lambda, ValueNotIgnorable, F#	PerceivedValue	Survey, ProfessionalProgrammers
S99	McEwan et al. 2010	LanguageComparison, VBA, C++	RuntimePerformance	ProgramRewrite, ResearcherParticipates
S100	McIver 2000	LanguageComparison, GRAIL, LOGO	ErrorProneness	NonrandomizedControlledExperiment, BetweenSubjects, BeginningProgrammingStudents
S101	Miara et al. 1983	FeaturePresence, ProgramIndentation, Pascal	ProgramComprehension	NonrandomizedControlledExperiment, ProgrammingStudents

(continues)

TABLE 23 (continues)

Study	Citation	Design decision codes	Efficacy codes	Method codes
S102	Millstein 2004; Millstein et al. 2009	FeaturePresence, PredicateDispatch, LanguageComparison, JPred, Java	LinesOfCodeComparison, ProgramTranslationEffort, ProgramQuality, Reusability	ProgramRewrite, ResearcherParticipates
S103	Mortensen et al. 2012	ParadigmComparison, AOP, OOP, LanguageComparison, AspectC++, C++	ProgramTranslationEffort, LinesOfCodeComparison	ProgramRewrite, CodeHistory
S104	Myers, Giuse, et al. 1992	LanguageComparison, CommonLisp, ObjectPascal, C++	ProgrammingEffort, LinesOfCodeComparison	LanguageShootout, ProfessionalProgrammers
S105	Myrtveit and Stensrud 2008	LanguageComparison, C, C++	ProgrammingEffort	MetricsCollectionAnalysis
S106	Nanz et al. 2010; Nanz et al. 2011b,a	LanguageComparison, Eiffel, Java, FeatureDesign, Concurrency	ProgramComprehension, ErrorProneness, DebuggingEffort	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S107	Necula et al. 2005	LanguageComparison, CCured, C	RuntimePerformance, AnnotationOverhead	ProgramRewrite, ResearcherParticipates
S108	Norcio 1982	FeaturePresence, ProgramIndentation	ClozeTestPerformance	RandomizedControlledExperiment, ProgrammingStudents, BetweenSubjects
S110	Nystrom et al. 2006	FeaturePresence, NestedIntersection, FeatureDesign, StaticTyping, LanguageComparison, J&, Java	ProgramTranslationEffort	ProgramRewrite, ResearcherParticipates
S111	Nyström et al. 2007	LanguageComparison, GdH, Erlang, C++/CORBA, C++/UDP	LinesOfCodeComparison, ProgrammingEffort	ProgramRewrite, ResearcherParticipates
S114	Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011	FeatureDesign, STM, MemoryLocking	LinesOfCodeComparison, ProgrammingEffort	RandomizedControlledExperiment, BetweenSubjects, AdvancedProgrammingStudents
S115	Pankratius, Schmidt, et al. 2012	LanguageComparison, Scala, Java	ProgrammingEffort, LinesOfCodeComparison	RandomizedControlledExperiment, AdvancedProgrammingStudents, ProfessionalProgrammers, WithinSubjects
S116	Patel and Gilbert 2008	LanguageComparison, C+MPI, UPC	ErrorProneness, LinesOfCodeComparison, ProgrammingEffort, RuntimePerformance	ReAnalysis, NonrandomizedControlledExperiment, AdvancedProgrammingStudents, WithinSubjects
S117	Patterson 1981	LanguageComparison, TRANSLANG, STRUM, Microprogramming	LinesOfCodeComparison, RuntimePerformance	LanguageShootout, ResearcherParticipates
S118	Perrott et al. 1980	LanguageComparison, Fortran, SIMONE	ProgramTranslationEffort, DebuggingEffort, ErrorProneness	ProgramRewrite, ResearcherParticipates
S119	Poletto et al. 1999	FeaturePresence, RuntimeCodeGeneration, LanguageComparison, 'C, C	RuntimePerformance	BenchmarkPrograms
S120	Prechelt and Tichy 1996, 1998	FeaturePresence, ArgumentTypeChecking, FeatureDesign, StaticTyping, C	ErrorProneness	RandomizedControlledExperiment, WithinSubjects, ProfessionalProgrammers
S121	Prechelt 2000; Prechelt 2003	ParadigmComparison, ScriptingParadigm, SystemProgrammingParadigm, LanguageComparison, C, C++, Java, Perl, Python, Rexx, Tcl	LinesOfCodeComparison, ErrorProneness, ProgrammingEffort	LanguageShootout
S122	Prechelt, Unger, et al. 2003; Unger and Prechelt 1998	FeaturePresence, ClassInheritance	ErrorProneness, MaintenanceEffort	SimulatedMaintenance, RandomizedControlledExperiment, ProgrammingStudents, BetweenSubjects
S123	Przybyłek 2011	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java	Modularity	CorpusAnalysis
S124	Qi and Myers 2010	FeaturePresence, FamilySharing, FeatureDesign, StaticTyping, LanguageComparison, J&h, Java	LinesOfCodeComparison	ProgramRewrite, ResearcherParticipates
S125	Ramalingam and Wiedenbeck 1997	ParadigmComparison, ProceduralParadigm, OOP, C++	ProgramComprehension	NonrandomizedControlledExperiment, WithinSubjects, BeginningProgrammingStudents
S127	Rossbach et al. 2009, 2010	FeatureDesign, MemoryLocking, STM	ProgrammingEffort, ErrorProneness	RandomizedControlledExperiment, AdvancedProgrammingStudents, WithinSubjects
S128	Saal and Weiss 1977	LanguageComparison, APL, Fortran	FeaturePrevalence	CorpusAnalysis, HistoricalControl
S130	Sawadpong et al. 2012	FeaturePresence, ExceptionHandling	ErrorProneness	CorpusAnalysis, BugHistory
S131	Scholte et al. 2012	FeaturePresence, StaticTyping, SecurityIssuePrevention	SecurityVulnerabilityProneness	CorpusAnalysis
S132	Seixas et al. 2009	FeatureDesign, StaticTyping, DynamicTyping	SecurityVulnerabilityProneness	CorpusAnalysis, HistoricalControl
S134	Sheppard et al. 1979	StructuredProgramming, FeaturePresence, Comments, Fortran	ProgramComprehension, MaintenanceEffort	RandomizedControlledExperiment, ProfessionalProgrammers, WithinSubjects
S136	Shneiderman 1976; Shneiderman and Mayer 1979	FeatureDesign, Conditionals	ProgramComprehension	NonrandomizedControlledExperiment, ProgrammingStudents, WithinSubjects
S137	Sime et al. 1973, 1999	FeatureDesign, Conditionals, JUMP, NEST	ProgrammingEffort, ErrorProneness	NonrandomizedControlledExperiment, NonProgrammers
S138	Sime et al. 1977	FeatureDesign, Conditionals, JUMP, NEST-BE, NEST-INE	ErrorProneness	NonrandomizedControlledExperiment, NonProgrammers
S140	Smith and Dunsmore 1982	StructuredProgramming, FeatureDesign, Conditionals, Loops, Fortran	ProgramComprehension	OtherExperiment, BeginningProgrammingStudents

(continues)

TABLE 23 (continues)

Study	Citation	Design decision codes	Efficacy codes	Method codes
S141	Soloway et al. 1983	FeatureDesign, Loops	ErrorProneness	NonrandomizedControlledExperiment, ProgrammingStudents, BetweenSubjects
S142	Stefik, Siebert, et al. 2011	LanguageComparison, Quorum, Randomo, Perl	ErrorProneness	RandomizedControlledExperiment, BetweenSubjects, NonProgrammers, ArtifactEncoding
S143	Stefik and Gellenbeck 2011	FeatureDesign, Loops, BooleanQueries, AssignmentSyntax, CallSyntax, StringLiteralSyntax, StringConcatenationSyntax, Conditionals	PerceivedIntuitivity	Survey, NonProgrammers, AdvancedProgrammingStudents
S144	Stuchlik and Hanenberg 2011	FeatureDesign, StaticTyping, DynamicTyping, TypeCasting, LanguageComparison, Java, Groovy	ProgrammingEffort	RandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents
S145	Taveira et al. 2009	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java	Reusability	ProgramRewrite, ResearcherParticipates
S146	Tenny 1985	FeaturePresence, NestedSubroutines, Comments	ProgramComprehension	NonrandomizedControlledExperiment, AdvancedProgrammingStudents, BetweenSubjects
S147	Thies and Amarasinghe 2010	FeatureDesign, StreamProgramming, StreamIt	FeaturePrevalence	CorpusAnalysis
S148	Tobin-Hochstadt and Felleisen 2008	LanguageComparison, TypedScheme, PLTScheme	ProgramTranslationEffort, LinesOfCodeComparison	ProgramRewrite, ResearcherParticipates
S149	Tonella and Ceccato 2005	ParadigmComparison, AOP, OOP	MaintenanceEffort, ProgramComprehension, LinesOfCodeComparison, Modularity	SimulatedMaintenance, RandomizedControlledExperiment, WithinSubjects, AdvancedProgrammingStudents, ProfessionalProgrammers
S150	Valente et al. 2010	ParadigmComparison, AOP, OOP	ProgramQuality	QualityMetrics, ProgramPairAnalysis
S151	Vessey and Weber 1984a	FeatureDesign, Conditionals, FeaturePresence, ProgramIndentation, JUMP-M, NEST, NEST-BE, NEST-INE	ProgrammingEffort, ErrorProneness	NonrandomizedControlledExperiment, BetweenSubjects, NonProgrammers
S153	Volos et al. 2009	FeatureDesign, STM, MemoryLocking, NestedParallelism	RuntimePerformance	BenchmarkPrograms
S154	Walker, Bamassad, et al. 1998; Walker, Baniassad, et al. 1999	ParadigmComparison, AOP, OOP, LanguageComparison, AspectJ, Java	DebuggingEffort, MaintenanceEffort	NonrandomizedControlledExperiment, AdvancedProgrammingStudents, ProfessionalProgrammers, BetweenSubjects, ProgrammerObservation
S155	Walker, Lamere, et al. 2002	LanguageComparison, Java, C	RuntimePerformance	ProgramRewrite, ResearcherParticipates
S156	Weimer and Necula 2008	FeaturePresence, CompensationStacks	ProgramTranslationEffort, RuntimePerformance	ProgramRewrite, ResearcherParticipates
S157	Westbrook et al. 2012	FeaturePresence, PermissionTypes, FeatureDesign, SharedMemoryCommunication, LanguageComparison, HJ, HJp	ProgramTranslationEffort	ProgramRewrite, ResearcherParticipates
S158	Wiedenbeck and Ramalingam 1999	ParadigmComparison, OOP, ProceduralParadigm, C++	ProgramComprehension	NonrandomizedControlledExperiment, BeginningProgrammingStudents, BetweenSubjects
S159	Wiedenbeck, Ramalingam, et al. 1999	ParadigmComparison, OOP, ProceduralParadigm, LanguageComparison, C++, Pascal	ProgramComprehension	NonrandomizedControlledExperiment, BeginningProgrammingStudents, BetweenSubjects

TABLE 24 Secondary studies and their assigned method codes

Study	Citation	Method codes
S5	Arblaster 1982	NarrativeReview
S15	Boehm-Davis 2002	NarrativeReview
S16	Briand et al. 1999	NarrativeReview
S32	Curtis 1982	NarrativeReview
S35	Deligiannis et al. 2002	NarrativeReview, SearchDatabasesSpecified, SearchTermsSpecified
S52	Furuta and Kemp 1979	NarrativeReview
S53	Fyfe 1997b,a	NarrativeReview
S61	Green 1980	NarrativeReview
S73	Hoc 1983	NarrativeReview
S81	Johnson 2002	NarrativeReview
S87	Laughery and Laughery 1985	NarrativeReview
S112	Pane and Myers 2000	NarrativeReview
S113	Pane and Myers 2006	NarrativeReview
S126	Roberts 1995	NarrativeReview
S129	Sadowski and Shewmaker 2010	NarrativeReview
S133	Sheil 1981	NarrativeReview
S135	Shneiderman 1975	NarrativeReview
S139	Sime, Arblaster, et al. 1977	NarrativeReview
S152	Vessey and Weber 1984b	NarrativeReview

APPENDIX 4 INCLUDED SECONDARY STUDIES

TABLE 25 Reports included in this mapping study that are also considered by included secondary studies

Study	Citation	Reports considered
S5a	Arblaster 1982	Green 1977 (S60), Green 1980 (S61), Sime et al. 1973 (S137), Sime, Arblaster, et al. 1977 (S139)
S15a	Boehm-Davis 2002	Ramalingam and Wiedenbeck 1997 (S125), Wiedenbeck and Ramalingam 1999 (S158)
S15b	Boehm-Davis 2002	Norcio 1982 (S108)
S16a	Briand et al. 1999	Daly et al. 1996 (S33)
S32a	Curtis 1982	Green 1977 (S60), Sime, Arblaster, et al. 1977 (S139)
S32b	Curtis 1982	Gannon 1976 (S55)
S32c	Curtis 1982	Gannon 1977 (S56)
S35a	Deligiannis et al. 2002	Henry and Humphrey 1990 (S69)
S35b	Deligiannis et al. 2002	Lewis et al. 1991 (S90)
S35c	Deligiannis et al. 2002	Wiedenbeck, Ramalingam, et al. 1999 (S159)
S35d	Deligiannis et al. 2002	Cartwright 1998 (S19, S20), Daly et al. 1996 (S33), Harrison, Counsell, et al. 2000 (S68), Unger and Prechelt 1998 (S122)
S52a	Furuta and Kemp 1979	Green 1977 (S60), Sime et al. 1973 (S137), Sime et al. 1977 (S138)
S52c	Furuta and Kemp 1979	Gannon and Horning 1975b (S55), Gannon and Horning 1975a (S55), Gannon 1976 (S55)
S52d	Furuta and Kemp 1979	Gannon 1977 (S56)
S53a	Fyfe 1997b,a	Harrison, Smaraweera, et al. 1996 (S67)
S61b	Green 1980	Green 1977 (S60), Sime et al. 1977 (S138)
S73a	Hoc 1983	Green 1977 (S60), Green 1980 (S61), Sime et al. 1973 (S137), Sime et al. 1977 (S138)
S73b	Hoc 1983	Shneiderman 1976 (S136)
S73c	Hoc 1983	Embley 1978 (S43)
S73d	Hoc 1983	Lucas and Kaplan 1976 (S93)
S73e	Hoc 1983	Sheppard et al. 1979 (S134)
S81a	Johnson 2002	Lewis et al. 1992 (S90)
S81b	Johnson 2002	Harrison, Smaraweera, et al. 1996 (S67)
S87a	Laughery and Laughery 1985	Green 1980 (S61), Sime et al. 1977 (S138)
S126a	Roberts 1995	Soloway et al. 1983 (S141)
S129a	Sadowski and Shewmaker 2010	Rossbach et al. 2010 (S127)
S129b	Sadowski and Shewmaker 2010	Ebcioğlu et al. 2006 (S42)
S129c	Sadowski and Shewmaker 2010	Luff 2009 (S94)
S129e	Sadowski and Shewmaker 2010	Hochstein, Basili, et al. 2008 (S74)
S133a	Sheil 1981	Green 1977 (S60), Sime et al. 1973 (S137), Sime et al. 1977 (S138)
S133b	Sheil 1981	Lucas and Kaplan 1976 (S93)
S133c	Sheil 1981	Gannon 1977 (S56)
S133d	Sheil 1981	Gannon and Horning 1975b (S55), Gannon 1976 (S55)
S135a	Shneiderman 1975	Sime et al. 1973 (S137)
S139a	Sime, Arblaster, et al. 1977	Green 1977 (S60), Sime et al. 1973 (S137), Sime et al. 1977 (S138)
S152a	Vessey and Weber 1984b	Green 1977 (S60), Sime et al. 1973 (S137), Sime et al. 1977 (S138)
S152b	Vessey and Weber 1984b	Lucas and Kaplan 1976 (S93)

TABLE 26 Reports considered by included secondary studies that have not been included in this mapping study

Study	Citation	Reports considered
S5a	Arblaster 1982	A. T. Arblaster & M. E. Sime & T. R. G. Green (1979): Jumping to some purpose. Computer Journal 22 (2), 105-109. (No recorded exclusion decision.)
S5a	Arblaster 1982	A. T. Arblaster (1977): Some measures of information about program states. International Computing Symposium 1977. (No recorded exclusion decision.)
S5a	Arblaster 1982	M. E. Sime, A. T. Arblaster & T. R. G. Green (1977): Reducing programming errors in nested conditionals by prescribing a writing procedure. International Journal of Man-Machine Studies 9 (1). Pages 119-126. doi:10.1016/S0020-7373(77)80046-1 (Excluded from this mapping study.)
S5a	Arblaster 1982	T. R. G. Green (1980): Programming as a cognitive activity. In Smith & Green (eds): Human Interaction with Computers. London: Academic Press. (No recorded exclusion decision.)
S15a	Boehm-Davis 2002	C. L. Corritore & S. Wiedenbeck (1999): Mental representations of expert procedural and object-oriented programmers in a software maintenance task. International Journal of Human-Computer Studies 50, 61-84. (No recorded exclusion decision.)
S15a	Boehm-Davis 2002	Françoise Détienne (1997): Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. Interacting with Computers 9 (1). Pages 47-72. doi: 10.1016/S0953-5438(97)00006-4 (Excluded from this mapping study.)
S16b	Briand et al. 1999	S. Benlarbi and W. L. Melo (1999): Polymorphism measures for early risk detection. In Proc. ICSE'99. (No recorded exclusion decision.)
S32a	Curtis 1982	T. R. G. Green, M. E. Sime & M. J. Fitter (1980): The problem the programmer faces. Ergonomics 23 (9), p. 893-907. (No recorded exclusion decision.)
S52b	Furuta and Kemp 1979	R. E. Mayer (1976): Comprehension as affected by structure of problem representation. Memory & Cognition 4(3), p. 249-255. (No recorded exclusion decision.)
S52d	Furuta and Kemp 1979	J. D. Gannon: Data types and programming reliability: Some preliminary evidence. Presented at the Symposium on Computer Software Engineering. Polytechnic Institute of New York (April 20–22, 1976) (No recorded exclusion decision.)
S61a	Green 1980	V. G. Richards & T. R. G. Green & J. Manton (1979): What Does Problem Representation Affect: Chunk Size, Memory Load, or Mental Process? Memo no. 319, MRC Social and Applied Psychology Unit, University of Sheffield. (No recorded exclusion decision.)
S61b	Green 1980	A. T. Arblaster & M. E. Sime & T. R. G. Green (1979): Jumping to some purpose. The Computer Journal 22, p. 105-109. (This is a review of other studies.) (No recorded exclusion decision.)

(continues)

TABLE 26 (continues)

Study	Citation	Reports considered
S73e	Hoc 1983	L. Weissman (1974): Psychological complexity of computer programs and experimental methodology. ACM SIGPLAN Notices 9. (No recorded exclusion decision.)
S87a	Laughery and Laughery 1985	A. T. Arblaster & M. E. Sine & T. R. G. Green (1975): Jumping to some purpose. Computer Journal 22. (No recorded exclusion decision.)
S87a	Laughery and Laughery 1985	M. E. Sime, A. T. Arblaster & T. R. G. Green (1977): Reducing programming errors in nested conditionals by prescribing a writing procedure. International Journal of Man-Machine Studies 9 (1). Pages 119-126. doi:10.1016/S0020-7373(77) 80046-1 (Excluded from this mapping study.)
S87a	Laughery and Laughery 1985	T. R. G. Green & M. E. Sine & M. Fitter (1975): Behavioral Experiments on Programming Languages. Memo 66. MRC Social and Applied Psychology, University of Sheffield, England. (No recorded exclusion decision.)
S112a	Pane and Myers 2000	Pane, J. F., & Myers, B. A. (2000). Tabular and Textual Methods for Selecting Objects from a Group. submitted for publication, http://www.cs.cmu.edu/~pane/Study3.html. That URL gives the following citation: J.F. Pane and B.A. Myers, "Tabular and Textual Methods for Selecting Objects from a Group," Proceedings of VL 2000: IEEE International Symposium on Visual Languages, Seattle, WA: IEEE Computer Society, September 10-13 2000, pp. 157-164. (No recorded exclusion decision.)
S113a	Pane and Myers 2006	J. F. Pane & B. A. Myers (2002): The impact of human-centered features on the usability of a programming system for children. In CHI 2002. (No recorded exclusion decision.)
S126b	Roberts 1995	Henry Shapiro (1980): "The results of an informal study to evaluate the effectiveness of teaching structured programming". SIGCSE Bulletin, December 1980. (No recorded exclusion decision.)
S129d	Sadowski and Shewmaker 2010	L. Hochstein, J. Carver, F. Shull, S. Asgari & V. Basili (2005): Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers. In Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference. Pages 35. doi:10.1109/SC.2005.53 (Excluded from this mapping study.)
S133e	Sheil 1981	"LOVE, T. "Relating individual differences in computer programming performance to human information processing abilities," Ph.D. dissertation, Univ. Washington, 1977." (p. 120) (No recorded exclusion decision.)
S133e	Sheil 1981	"Shneiderman and McKay (reported in SHNE80)" (p. 109), where SHNE80 is "SHNEIDERMAN,B. Software psychology, Winthrop, Cambridge, Mass., 1980" (p. 120) (No recorded exclusion decision.)
S133e	Sheil 1981	"WEISSMAN, L. "A methodology for studying the psychological complexity of computer programs," Ph.D. dissertation, Univ. Toronto, Canada, 1974." (p. 120) (No recorded exclusion decision.)
S139b	Sime, Arblaster, et al. 1977	R. E. Mayer (1976): Comprehensions as affected by structure of problem representation. Mem. Cog. 4, 249-255. (No recorded exclusion decision.)

APPENDIX 5 EXCLUDED PUBLICATIONS

The following publications were recorded during searches as not being obviously irrelevant but were subsequently excluded from this mapping study.

Exclusion can happen two ways: either the answer to both of the selection questions Q1 and Q2 (see page 64) was negative, or one of the selection questions Q3–Q7 was answered negatively. The questions whose negative answer caused the exclusion are listed for each excluded publication.

For most decisions, verbal explanations have been recorded. They are reproduced below, after the exclusion reason. Each explanation is preceded by a tag indicating the phase in which the explanation was written, and the author of the explanation. Tags starting with "II" indicate that full text was not yet retrieved at the time of the explanation; tags starting with "III" indicate that full text was considered. Explanations recorded during selection evaluation have been given tags starting with either "sel-1" or "sel-2". Post-hoc exclusions are indicated by a "posthoc" tag. Most tags also indicate the explanation's author: I am ".ajk", Ville Tirronen's explanations (which may have been paraphrased) are tagged ".tirronen". If no author is indicated, I should be presumed to have written the explanation.

1. M. Abadi, L. Cardelli, B. Pierce & G. Plotkin (1989): Dynamic typing in a statically-typed language. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 213-227. doi:10.1145/75277.75296 Exclusion reasons: Q5 [II.ajk]Elaboration of a language construct with theoretical and compiler-development evaluation only, based on the abstract.
2. Martín Abadi, Luca Cardelli, Benjamin Pierce & Gordon Plotkin (1991): Dynamic typing in a statically typed language. ACM Transactions on Programming Languages and Systems 13 (2). Pages 237-268. doi:10.1145/103135.103138 Exclusion reasons: Q5 [II.ajk]Formal type-theoretic work.
3. Martín Abadi, Luca Cardelli & Pierre-Louis Curien (1993): Formal parametric polymorphism. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 157-170. doi:10.1145/158511.158622 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work
4. Martín Abadi & Luca Cardelli (1995): On Subtyping and Matching. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 145-167. doi:10.1007/3-540-49538-X_8 Exclusion reasons: Q1–2 [II.ajk]Type-theoretical work.
5. Martín Abadi & Luca Cardelli (1996): On subtyping and matching. ACM Transactions on Programming Languages and Systems 18 (4). Pages 401-423. doi:10.1145/233561.233563 Exclusion reasons: Q5 [II.ajk]Formal type theoretical development.
6. Martín Abadi (1998): Protection in Programming-Language Translations: Mobile Object Systems. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 581. doi:10.1007/3-540-49255-0_70 Exclusion reasons: Q5 [III.ajk]There is a fuller paper at doi:10.1007/BFb0055109, but neither this nor that aspires to empiricity.
7. Martín Abadi, Cédric Fournet & Georges Gonthier (2000): Authentication primitives and their compilation. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 302-315. doi:10.1145/325694.325734 Exclusion reasons: Q1–2 [II.ajk]Formal development.
8. Martín Abadi & Cédric Fournet (2001): Mobile values, new names, and secure communication. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 104-115. doi:10.1145/360204.360213 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical study, no language design issue.
9. Martín Abadi, Andrew Birrell, Tim Harris & Michael Isard (2011): Semantics of transactional memory and automatic mutual exclusion. ACM Transactions on Programming Languages and Systems 33 (1). Pages 2:1–2:50. doi:10.1145/1889997.1889999 Exclusion reasons: Q5 [II.ajk]Theoretical work, looks like.
10. A.S. Abbas, W. Jeberson & VV Klinsega (2012): A Literature Review and Classification of Selected Software Engineering Researches. International Journal of Engineering and Technology 2 (7). http://iet-journals.org/archive/2012/july_vol_2_no_7/7565991339399989.pdf Exclusion reasons: Q1–2 [III.ajk]This is a mapping study.
11. Russell J. Abbott (1983): Program design by informal English descriptions. Communications of the ACM 26 (11). Pages 882-894. doi:10.1145/182.358441 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
12. Russell J. Abbott (1987): Knowledge abstraction. Communications of the ACM 30 (8). Pages 664-671. doi:10.1145/27651.27652 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
13. Soufyane Aboubekr, Gwenaël Delaval & Éric Rutten (2009): A programming language for adaptation control: case study. SIGBED Review 6 (3). Article 11. Pages 11:1–11:5. doi:10.1145/1851340.1851353 Exclusion reasons: Q5 [III.ajk]This article explores the implications of a particular artefact, and thus does not aspire to empiricality.
14. S. Abrahao, E. Insfran, C. Gravino & G. Scanniello (2009): On the effectiveness of dynamic modeling in UML: Results from an external replication. In Third international symposium on Empirical Software Engineering and Measurement ESEM 2009. Pages 468-472. doi:10.1109/ESEM.2009.5316004 Exclusion reasons: Q1–2 [II.ajk]No PL relevance.
15. Umut A. Acar, Guy E. Blelloch & Robert Harper (2002): Adaptive functional programming. In Proc. 29th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 247-259. doi:10.1145/503272.503296 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
16. Umut A. Acar, Guy E. Blelloch & Robert Harper (2003): Selective memoization. In Proc. 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 14-25. doi:10.1145/604131.604133 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
17. Umut A. Acar, Guy E. Blelloch & Robert Harper (2006): Adaptive functional programming. ACM Transactions on Programming Languages and Systems 28 (6). Pages 990-1034. doi:10.1145/1186632.1186634 Exclusion reasons: Q5 [II.ajk]Formal theoretical and implementation study.
18. Umut A. Acar, Amal Ahmed & Matthias Blume (2008): Imperative self-adjusting computation. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 309-322. doi:10.1145/1328438.1328476 Exclusion reasons: Q1–2 [II.ajk]Language exposition and theoretical work.
19. Umut A. Acar, Guy E. Blelloch, Matthias Blume, Robert Harper & Kanat Tangwongsan (2009): An experimental analysis of self-adjusting computation. ACM Transactions on Programming Languages and Systems 32 (1). doi:10.1145/1596527.1596530 Exclusion reasons: Q1–2 [III.ajk]This article describes an embedded language for describing self-adjusting computations, and reports on a benchmarking study comparing ordinary programs and their self-adjusting counterparts with respect to speed under data loads that appear to favor self-adjusting computation. I doubt it has any relevance to programmer-experienced efficacy of the language design.

20. Umut A. Acar, Arthur Charguéraud & Mike Rainey (2011): Oracle scheduling: controlling granularity in implicitly parallel languages. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 499-518. doi:10.1145/2048066.2048106 Exclusion reasons: Q1–2 [III.ajk]The empirical evaluation concerns implementation efficiency only.
21. Eldridge S. Adams, Jr. (1958): Simple automatic coding systems. Communications of the ACM 1 (7). Pages 5-9. doi:10.1145/368873.368884 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
22. Michael D. Adams, Andrew W. Keep, Jan Midtgaard, Matthew Might, Arun Chauhan & R. Kent Dybvig (2011): Flow-sensitive type recovery in linear-log time. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 483-498. doi:10.1145/2048066.2048105 Exclusion reasons: Q1–2 [II.ajk]This article deals with an implementation technique only.
23. T. R. ADDIS & J. J. TOWNSEND ADDIS (2002): An introduction to clarity: a schematic functional language for managing the design of complex systems. International Journal of Human-Computer Studies 56 (4). Pages 331-374. doi:10.1006/ijhc.2002.0528 Exclusion reasons: Q1–2 [II.ajk]The language under consideration is not textual.
24. Luiz Marques Afonso, Renato F. de G. Cerqueira & Clarisse Sieckenius de Souza (2012): Evaluating Application Programming Interfaces as Communication Artefacts. In PPIG 2012. Exclusion reasons: Q1–2 [II.ajk]No language design issues.
25. Edward E. Aftandilian, Samuel Z. Guyer, Martin Vechev & Eran Yahav (2011): Asynchronous assertions. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 275-288. doi:10.1145/2048066.2048090 Exclusion reasons: Q5 [III.ajk]This article has empirical evaluation of performance only.
26. Ritu Agarwal, Atish P. Sinha & Mohan Tanniru (1996): The role of prior experience and task characteristics in object-oriented modeling: an empirical study. International Journal of Human-Computer Studies 45 (6). Pages 639-667. doi:10.1006/ijhc.1996.0072 Exclusion reasons: Q1–2 [III.ajk]This article studies modeling, not programming.
27. R. Agarwal, P. De & A.P. Sinha (1999): Comprehending object and process models: an empirical study. Software Engineering, IEEE Transactions on 25 (4). Pages 541-556. doi:10.1109/32.799953 Exclusion reasons: Q1–2 [III.ajk]This article does not does not discuss programming language matters.
28. Ole Agesen (1995): The Cartesian Product Algorithm: Simple and Precise Type Inference of Parametric Polymorphism. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 2-26. doi:10.1007/3-540-49538-X_2 Exclusion reasons: Q1–2 [II.ajk]Implementation technique development.
29. Amal Ahmed, Robert Bruce Findler, Jeremy G. Siek & Philip Wadler (2011): Blame for all. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 201-214. doi:10.1145/1926385.1926409 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
30. Mehmet Aksit & Lodewijk Bergmans (1992): Obstacles in object-oriented software development. In conference proceedings on Object-oriented programming systems, languages, and applications. New York, NY, USA: ACM. Pages 341-358. doi:10.1145/141936.141965 http://purl.utwente.nl/publications/64957 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
31. Mehmet Akşit, Lodewijk Bergmans & Sinan Vural (1992): An object-oriented language-database integration model: The composition-filters approach. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 372-395. doi:10.1007/BFb0053047 Exclusion reasons: Q1–2 [II.ajk]Feature exposition.
32. M. M. Al-Jarrah & I. S. Torsun (1979): An empirical analysis of COBOL programs. Software: Practice and Experience 9 (5). Pages 341-359. doi:10.1002/spe.4380090502 Exclusion reasons: Q1–2 [II.ajk]No comparison.
33. S. Alagić, R. Sunderraman & R. Bagai (1994): Declarative object-oriented programming: Inheritance, subtyping and prototyping. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 236-259. doi:10.1007/BFb0052186 Exclusion reasons: Q5 [II.ajk]Exposition, implementation.
34. Suad Alagić, Jose Solorzano & David Gitchell (1998): Orthogonal to the Java imperative. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 212-233. doi:10.1007/BFb0054093 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
35. Rola Alameh, Nico Zazworka & Jeffrey K. Hollingsworth (2007): Performance Measurement of Novice HPC Programmers Code. In Proceedings of the 3rd International Workshop on Software Engineering for High Performance Computing Applications. Washington, DC, USA: IEEE Computer Society. SE-HPC '07. doi:10.1109/SE-HPC.2007.4 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
36. Ahmed Alardawi, Babak Khazaei & Jawed Siddiqi (2011): The influence of class structure on program comprehension. In PPIG 2011. http://ppig.org/papers/23/25%20Alardawi.pdf Exclusion reasons: Q1–2 [III.ajk]This article studies program organization and does not seem to have any language design decision at stake,
37. Jonathan Aldrich, Vibha Sazawal, Craig Chambers & David Notkin (2003): Language Support for Connector Abstractions. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 74-102. doi:10.1007/978-3-540-45070-2_5 Exclusion reasons: Q5 Disagreement resolution result. [sel-2.kaijanaho]Analytical.
38. Jonathan Aldrich & Craig Chambers (2004): Ownership Domains: Separating Aliasing Policy from Mechanism. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 1-25. doi:10.1007/978-3-540-24851-4_1 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
39. Jonathan Aldrich (2005): Open Modules: Modular Reasoning About Advice. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 144-168. doi:10.1007/11531142_7 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
40. Jonathan Aldrich, Robert Bocchino, Ronald Garcia, Mark Hahnenberg, Manuel Mohr, Karl Naden, Darpan Saini, Sven Stork, Joshua Sunshine, Éric Tanter & Roger Wolff (2011): Plaid: a permission-based programming language. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 183-184. doi:10.1145/2048147.2048197 Exclusion reasons: Q5 [II.ajk]No empirical evaluation.
41. Andrei Alexandrescu & Konrad Lorincz (2002): ArchJava: An Evaluation. Student project report. http://archjava.fluid.cs.cmu.edu/papers/alexandrescu-lorincz-archjava.pdf Exclusion reasons: Q5 [III.ajk]This is an experience report and an analytical study.
42. Giora Alexandron, Michal Armoni & David Harel (2011): Programming with the user in mind. In PPIG 2011. http://ppig.org/papers/23/20%20Alexandron.pdf Exclusion reasons: Q1–2 [III.ajk]The language in question is diagrammatic, not textual.
43. Ghazi Alkhatib (1992): The maintenance problem of application software: An empirical analysis. Journal of Software Maintenance: Research and Practice 4 (2). Pages 83-104. doi:10.1002/smr.4360040203 Exclusion reasons: Q1–2 [III.ajk]This article describes a case study attempting to determine variables that affect maintenance load. It does not evaluate any language design decisions in any nontrivial sense.
44. Chris Allan, Pavel Avgustinov, Aske Simon Christensen, Laurie Hendren, Sascha Kuzins, Ondřej Lhoták, Oege de Moor, Damien Sereni, Ganesh Sittampalam & Julian Tibble (2005): Adding trace matching with free variables to AspectJ. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 345-364. doi:10.1145/1094811.1094839 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate its design choices empirically; there's one section that might look like it does but as the conclusion is "[t]hese results demonstrate the feasibility of our approach"; not efficacy but feasibility.
45. Carl Martin Allwood (1986): Novices on the computer: a review of the literature. International Journal of Man-Machine Studies 25 (6). Pages 633-658. doi:10.1016/S0020-7373(86)80079-7 Exclusion reasons: Q1–2 [II.ajk]Not a PL design issue.
46. Carl Martin Allwood & Carl-Gustav Björhag (1990): Novices' debugging when programming in Pascal. International Journal of Man-Machine Studies 33 (6). Pages 707-724. doi:10.1016/S0020-7373(05)80070-7 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
47. Jay Almarode (1991): Issues in the design and implementation of a schema designer for an OODBMS. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 200-218. doi:10.1007/BFb0057023 Exclusion reasons: Q5 [III.ajk]This article does not evaluate language design decisions.
48. Paulo Sérgio Almeida (1997): Balloon types: Controlling sharing of state in data types. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 32-59. doi:10.1007/BFb0053373 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
49. Johan Per Fredrik Almqvist (2006): Replication of controlled experiments in empirical software engineering-A survey. . Master's thesis. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.117.9777&rep=rep1&type=pdf Exclusion reasons: Q7 [III.ajk]This Master's Thesis does not discuss empirical evidence in the primary studies under review.
50. Bowen Alpern, C. R. Attanasio, Anthony Cocchi, Derek Lieber, Stephen Smith, Ton Ngo, John J. Barton, Susan Flynn Hummel, Janice C. Sheperd & Mark Mergen (1999): Implementing jalapeño in Java. In OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 314-324. doi:10.1145/320384.320418 Exclusion reasons: Q1–2 [II.ajk]Implementation study.
51. Enrique Calderon Alzati (1966): AN EXPERIMENTAL PROGRAMMING LANGUAGE FOR TEACHING SYMBOLIC MANIPULATION.. at MOORE

SCHOOL OF ELECTRICAL ENGINEERING PHILADELPHIA PA. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0800050 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

52. H. Aman (2012): An Empirical Analysis on Fault-Proneness of Well-Commented Modules. In Empirical Software Engineering in Practice (IWESEP), 2012 Fourth International Workshop on. Pages 3-9. doi:10.1109/IWESEP.2012.12 Exclusion reasons: Q1–2 [II.ajk]Comment usage patterns are not a language design issue.
53. Pierre America & Frank van der Linden (1990): A parallel object-oriented language with inheritance and subtyping. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 161-168. doi:10.1145/97945.97966 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
54. Davide Ancona, Giovanni Lagorio & Elena Zucca (2000): Jam - A Smooth Extension of Java with Mixins. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 154-178. doi:10.1007/3-540-45102-1_8 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
55. Davide Ancona & Elena Zucca (2001): True Modules for Java-like Languages. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 354-380. doi:10.1007/3-540-45337-7_19 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
56. Davide Ancona, Giovanni Lagorio & Elena Zucca (2003): Jam - designing a Java extension with mixins. ACM Transactions on Programming Languages and Systems 25 (5). Pages 641-712. doi:10.1145/937563.937567 Exclusion reasons: Q1–2 [II.ajk]Exposition of a new language feature.
57. Davide Ancona, Giovanni Lagorio & Elena Zucca (2006): Flexible Type-Safe Linking of Components for Java-Like Languages. Volume 4228.In Lightfoot, David and Szyperski, Clemens (ed.) Modular Programming Languages.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 136-154. doi:10.1007/11860990_10 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
58. Bente C. D. Anda, Dag I. K. Sjoberg & Audris Mockus (2009): Variability and Reproducibility in Software Engineering: A Study of Four Companies that Developed the Same System. IEEE Transactions on Software Engineering 35 (3). Pages 407-429. doi:10.1109/TSE.2008.89 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design issues.
59. Bruce Anderson (1980): Programming in the home of the future. International Journal of Man-Machine Studies 12 (4). Pages 341-365. doi:10.1016/S0020-7373(80)80020-4 Exclusion reasons: Q1–2 [II.ajk]No language design issue.
60. Christopher Anderson, Paola Giannini & Sophia Drossopoulou (2005): Towards Type Inference for JavaScript. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 428-452. doi:10.1007/11531142_19 Exclusion reasons: Q1–2 [II.ajk]Theoretical and implementation work.
61. Zachary Anderson & David Gay (2011): Composable, nestable, pessimistic atomic statements. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 865-884. doi:10.1145/2048066.2048132 Exclusion reasons: Q1–2 [III.ajk]Evaluation focuses solely on implementation performance.
62. Zachary Anderson (2012): Efficiently combining parallel software using fine-grained, language-level, hierarchical resource management policies. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 717-736. doi:10.1145/2384616.2384669 Exclusion reasons: Q5 [III.ajk]The evaluation is analytic in nature (demonstrating that this can work reasonably well).
63. Gregory R. Andrews (1981): Synchronizing Resources. ACM Transactions on Programming Languages and Systems 3 (4). Pages 405-430. doi:10.1145/357146.357149 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
64. Gregory R. Andrews, Michael Coffin, Irving Elshoff, Kelvin Nilson, Gregg Townsend, Ronald A. Olsson & Titus Purdin (1988): An overview of the SR language and implementation. ACM Transactions on Programming Languages and Systems 10 (1). Pages 51-86. doi:10.1145/42192.42324 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
65. Nicos Angelopoulos & James Cussens (2003): Prolog Issues and Experimental Results of an MCMC Algorithm. Volume 2543.In Bartenstein, Oskar and Geske, Ulrich and Hannebauer, Markus and Yoshie, Osamu (ed.) Web Knowledge Management and Decision Support.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 186-196. doi:10.1007/3-540-36524-9_15 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a programming language design decision.
66. Ed Anson (1987): A generalized iterative construct and its semantics. ACM Transactions on Programming Languages and Systems 9 (4). Pages 567-581. doi:10.1145/29873.30391 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
67. Sven Apel, Christian Kastner & Salvador Trujillo (2007): On the Necessity of Empirical Studies in the Assessment of Modularization Mechanisms for Crosscutting Concerns. In Proceedings of the First International Workshop on Assessment of Contemporary Modularization Techniques. Pages 1-7. doi:10.1109/ACOM.2007.7 Exclusion reasons: Q1–2 [III.ajk]This article argues for, and proposes, an empirical study design; it does not actually report such a study.
68. Maria Virginia Aponte (1993): Extending record typing to type parametric modules with sharing. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 465-478. doi:10.1145/158511.158704 Exclusion reasons: Q5 [II.ajk]Formal type-theoretical study.
69. W. F. Appelbe & A. P. Ravn (1984): Encapsulation constructs in systems programming languages. ACM Transactions on Programming Languages and Systems 6 (2). Pages 129-158. doi:10.1145/2993.69615 Exclusion reasons: Q5 [III.ajk]This analytical article does not aspire to empiricity.
70. Krzysztof R. Apt & Andrea Schaerf (1997): Search and imperative programming. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 67-79. doi:10.1145/263699.263709 Exclusion reasons: Q5 [III.ajk]This is a constructive-analytical study of adding some logic programming features to an imperative programming language. It has no aspiration to empiricity.
71. Krzysztof R. Apt, Jacob Brunekreef, Vincent Partington & Andrea Schaerf (1998): Alma-O: an imperative language that supports declarative programming. ACM Transactions on Programming Languages and Systems 20 (5). Pages 1014-1066. doi:10.1145/293677.293679 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
72. W. Araujo, L.C. Briand & Y. Labiche (2011): On the Effectiveness of Contracts as Test Oracles in the Detection and Diagnosis of Race Conditions and Deadlocks in Concurrent Object-Oriented Software. In Empirical Software Engineering and Measurement (ESEM), 2011 International Symposium on. Pages 10-19. doi:10.1109/ESEM.2011.9 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions for efficacy.
73. B. Arden & R. Graham (1959): On GAT and the construction of translators. Communications of the ACM 2 (7). Pages 24-26. doi:10.1145/368370.368373 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
74. Bruce Arden, Bernard A. Galler & Robert M. Graham (1961): Letter to the editor: criticisms of ALGOL 60. Communications of the ACM 4 (7). Pages 309. doi:10.1145/366622.366625 Exclusion reasons: Q5 [III.ajk]This letter to the editor does not report an empirical study.
75. Erik Arisholm (2006): Empirical assessment of the impact of structural properties on the changeability of object-oriented software. Information and Software Technology 48 (11). Pages 1046-1055. doi:10.1016/j.infsof.2006.01.002 Exclusion reasons: Q1–2 Disagreement resolution result. [II.ajk]This article does not evaluate any language design decisions.
76. J.L. Armstrong & S.R. Virding (1990): ERLANG - an experimental telephony programming language. Volume 3.In Switching Symposium, 1990. XIII International. Pages 43-48. doi:10.1109/ISS.1990.765711 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
77. Joe Armstrong (2007): Erlang – Software for a Concurrent World. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 1. doi:10.1007/978-3-540-73589-2_1 Exclusion reasons: Q1–2 [III.ajk]This is a short abstract of a language exposition.
78. T. N. Arvanitis, M. J. Todd, A. J. Gibb & E. Orihashi (2001): Understanding students' problem-solving performance in the context of programming-in-the-small: an ethnographic field study. Volume 2.In Frontiers in Education Conference, 2001. 31st Annual. doi:10.1109/FIE.2001.963676 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
79. Arvind, Rishiyur S. Nikhil & Keshav K. Pingali (1989): I-structures: data structures for parallel computing. ACM Transactions on Programming Languages and Systems 11 (4). Pages 598-632. doi:10.1145/69558.69562 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.
80. E. A. Ashcroft & W. W. Wadge (1978): Clauses: scope structures and defined functions in Lucid. In Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 17-22. doi:10.1145/512760.512763 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
81. Aslan Askarov & Andrei Sabelfeld (2005): Security-Typed Languages for Implementation of Cryptographic Protocols: A Case Study. Volume 3679.In di Vimercati, Sabrina and Syverson, Paul and Gollmann, Dieter (ed.) Computer Security – ESORICS 2005.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 197-221. doi:10.1007/11555827_12 Exclusion reasons: Q5 [III.ajk]This article reports a study where the investigators wrote a program in Java and then translated it (more or less deterministically) into Jif. The study is not empirical by our definition, as there isn't much contingency in it (certainly not in the difference between the Java and Jif versions).

82. Medhat G. Assaad & Gary T. Leavens (2001): Alias-free parameters in C for better reasoning and optimization. Technical report. http://archives.cs.iastate.edu/documents/disk0/00/00/02/55/index.html Exclusion reasons: Q5 [III.ajk]This article reports, among other things, a study in which C/ACL is compared to C using a set of benchmark applications where the C code has been converted, presumably by the investigators, to C/ACL, with respect to execution speed. To the extent that this measures efficacy, it merely presents "ACL can do it" style results, and thus is not empirical (despite the use of measurements).
83. W. C. Athas (1985): XCPL: An experimental Concurrent Programming Language. CaltechCSTR:1985:5196-tr-85 at California Institute of Technology. http://resolver.caltech.edu/CaltechCSTR:1985.5196-tr-85 Exclusion reasons: Q5 [III.ajk]This report does not aspire to empiricity.
84. Russell Atkinson & Carl Hewitt (1977): Synchronization in actor systems. In Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 267-280. doi:10.1145/512950.512975 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
85. Malcolm P. Atkinson & Ronald Morrison (1985): Procedures as persistent data objects. ACM Transactions on Programming Languages and Systems 7 (4). Pages 539-559. doi:10.1145/4472.4477 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article does not aspire to empiricity.
86. Giuseppe Attardi, Cinzia Bonini, Maria Rosaria Boscotrecase, Tito Flagella & Mauro Gaspari (1989): Metalevel Programming in CLOS. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 243-256. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890243.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
87. Joshua Auerbach, David F. Bacon, Perry Cheng & Rodric Rabbah (2010): Lime: a Java-compatible and synthesizable language for heterogeneous architectures. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications. Pages 89-108. doi:10.1145/1869459.1869469 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
88. M. Auguston & A. Delgado (1997): The V Experimental Visual Programming Language. NASA University Research Centers Technical Advances in Education, Aeronautics, Space, Autonomy, Earth and Environment, 1. Pages 81–86. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.43.2074 Exclusion reasons: Q1–2 [III.ajk]This article discusses a visual language; such languages are excluded under our definition of programming languages.
89. Thomas H. Austin, Tim Disney & Cormac Flanagan (2011): Virtual values for language extension. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 921-938. doi:10.1145/2048066.2048136 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
90. Enis Avdicaušević, Marjan Mernik, Mitja Lenic & Viljem Zumer (2002): Experimental aspect-oriented language - AspectCOOL. In Proceedings of the 2002 ACM symposium on Applied computing. New York, NY, USA: ACM. Pages 943-947. doi:10.1145/508791.508974 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.
91. Enis Avdičaušević, Mitja Lenič, Marjan Mernik & Viljem Zumer (2001): AspectCOOL: an experiment in design and implementation of aspect-oriented language. SIGPLAN Notices 36. Pages 84-94. doi:10.1145/583960.583971 Exclusion reasons: Q1–2 [II.ajk]No comparison.
92. Hassan Aït-Kaci & Roger Nasr (1986): Logic and inheritance. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 219-228. doi:10.1145/512644.512664 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
93. Hassan Aït-Kaci & Andreas Podelski (1994): Functions as passive constraints in LIFE. ACM Transactions on Programming Languages and Systems 16 (4). Pages 1219-1318. doi:10.1145/183432.183526 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work.
94. Yair M. Babad & Jeffrey A. Hoffer (1984): Even no data has a value. Communications of the ACM 27 (8). Pages 748-756. doi:10.1145/358198.358204 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
95. B. Bacci, M. Danelutto, S. Orlando, S. Pelagatti & M. Vanneschi (1995): Summarising an experiment in parallel programming language design. Volume 919.In Hertzberger, Bob and Serazzi, Giuseppe (ed.) High-Performance Computing and Networking. Pages 7-13. doi:10.1007/BFb0046602 Exclusion reasons: Q7 [III.ajk]This article summarises research previously published by the authors. However, it does not discuss any empirical evidence such studies may have provided.
96. G. A. Bachelor, J. R. H. Dempster, D. E. Knuth & J. Speroni (1961): SMALGOL-61. Communications of the ACM 4 (11). Pages 499-502. doi:10.1145/366813.366843 Exclusion reasons: Q1–2 [III.ajk]This is a language specification, not a study.
97. R. J. R. Back & R. Kurki-Suonio (1988): Distributed cooperation with action systems. ACM Transactions on Programming Languages and Systems 10 (4). Pages 513-554. doi:10.1145/48022.48023 Exclusion reasons: Q5 [II.ajk]Theoretical study.
98. J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger & P. Naur (1963): Revised report on the algorithm language ALGOL 60. Communications of the ACM 6 (1). Pages 1-17. doi:10.1145/366193.366201 Exclusion reasons: Q1–2 [III.ajk]This language specification does not report an empirical study.
99. John Backus (1978): Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. Communications of the ACM 21 (8). Pages 613-641. doi:10.1145/359576.359579 Exclusion reasons: Q5 [III.ajk]This classic lecture does not report an empirical study.
100. Woongki Baek, Nathan Bronson, Christos Kozyrakis & Kunle Olukotun (2010): Making nested parallel transactions practical using lightweight hardware support. In Proceedings of the 24th ACM International Conference on Supercomputing. New York, NY, USA: ACM. ICS '10. Pages 61-71. doi:10.1145/1810085.1810097 Exclusion reasons: Q1–2 [III.ajk]This article evaluates an implementation technique.
101. Woongki Baek, Nathan Bronson, Christos Kozyrakis & Kunle Olukotun (2010): Implementing and evaluating nested parallel transactions in software transactional memory. In Proceedings of the 22nd ACM symposium on Parallelism in algorithms and architectures. New York, NY, USA: ACM. SPAA '10. Pages 253-262. doi:10.1145/1810479.1810528 Exclusion reasons: Q1–2 [III.ajk]This article deals with an implementation technique, not a language design decision.
102. Philip R. Bagley (1959): Proposal for a feasible programming system. Communications of the ACM 2 (8). Pages 7-10. doi:10.1145/368405.368410 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
103. Rajive Bagrodia (1989): Synchronization of asynchronous processes in CSP. ACM Transactions on Programming Languages and Systems 11 (4). Pages 585-597. doi:10.1145/69558.69561 Exclusion reasons: Q1–2 [II.ajk]Implementation technique
104. P. A. C. Bailes & L. H. Reeker (1980): An experimental applicative programming language for linguistics and string processing. In Proceedings of the 8th conference on Computational linguistics. Stroudsburg, PA, USA: Association for Computational Linguistics. Pages 520-525. doi:10.3115/990174.990270 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity
105. M. J. Bailey, M. P. Barnett & R. P. Futrelle (1963): Format-free input in FORTRAN. Communications of the ACM 6 (10). Pages 605-608. doi:10.1145/367651.367658 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
106. Michael J. Bailey (1964): More on "simple I/O" statements. Communications of the ACM 7 (5). Pages 314-315. doi:10.1145/364099.364327 Exclusion reasons: Q1–2 [III.ajk]This is a letter to the editor and does not report a study.
107. Jason Baker & Wilson C. Hsieh (2002): Maya: multiple-dispatch syntax extension in Java. In Proceedings of the ACM SIGPLAN 2002 Conference on Programming language design and implementation. New York, NY, USA: ACM. PLDI '02. Pages 270-281. doi:10.1145/512529.512562 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
108. A. G. Bakhmurov, V. V. Voevodin, N. N. Popova, R. L. Smelyanskii & A. V. Khanov (1991): Laplace — an experimental language for parallel programming of MHD-models in plasma physics. Computational Mathematics and Modeling 2 (4). Pages 467-471. doi:10.1007/BF01127968 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
109. T. Balafoutis, A. Paparrizou & K. Stergiou (2010): Experimental Evaluation of Branching Schemes for the CSP. In Proceedings of 2010 TRICS, 3rd Workshop on Techniques for Implementing Constraint Programming Systems. http://users.uowm.gr/kstergiou/TRICS2010.pdf Exclusion reasons: Q1–2 [III.ajk]This paper evaluates algorithms, not language designs.
110. Pierre F. Baldi, Cristina V. Lopes, Erik J. Linstead & Sushil K. Bajracharya (2008): A theory of aspects as latent topics. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 543-562. doi:10.1145/1449764.1449807 Exclusion reasons: Q1–2 [III.ajk]This article presents a method for mining aspects, and empirical results derived from such mining. It does not evaluate a language design decision.
111. Mark B. Ballard, David Maier & Allen Wirfs-Brock (1986): QUICKTALK: a Smalltalk-80 dialect for defining primitive methods. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 140-150. doi:10.1145/28697.28711 Exclusion reasons: Q5 [III.ajk]This article includes a short section comparing Quicktalk and Smalltalk versions of programs for speed. It The only contingency in this is the implementation; everything else is analytical.
112. R. M. Balzer & D. J. Farber (1969): APAREL—A parse-request language. Communications of the ACM 12 (11). Pages 624-631. doi:10.1145/363269.363606 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
113. Guruduth Banavar & Gary Lindstrom (1996): An application framework for module composition tools. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 91-113. doi:10.1007/BFb0053058 Exclusion reasons: Q1–2 [III.ajk]This paper does not evaluate any language design decisions.

114. Richard S. Bandat & Robert L. Wilkins (1967): An experimental general purpose compiler. In Proceedings of the April 18-20, 1967, spring joint computer conference. Pages 457-461. doi:10.1145/1465482.1465554 Exclusion reasons: Q1–2 [II.ajk]Discussion of implementation techniques.
115. Robert G. Bandes (1984): Constraining-unification and the programming language unicorn. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 106-110. doi:10.1145/800017.800521 Exclusion reasons: Q5 [III.ajk]This article has no empirical content.
116. Anindya Banerjee & David A. Naumann (2005): State Based Ownership, Reentrance, and Encapsulation. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 387-411. doi:10.1007/11531142_17 Exclusion reasons: Q5 [III.ajk]This theoretical article does not aspire to empiricity.
117. Anindya Banerjee, David A. Naumann & Stan Rosenberg (2008): Regional Logic for Local Reasoning about Global Invariants. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 387-411. doi:10.1007/978-3-540-70592-5_17 Exclusion reasons: Q5 [II.ajk]Theoretical study.
118. Elisa Baniassad & Sebastian Fleissner (2006): The geography of programming. In OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. Pages 510-520. doi:10.1145/1176617.1176625 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.
119. Joseph A. Bank, Andrew C. Myers & Barbara Liskov (1997): Parameterized types for Java. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 132-145. doi:10.1145/263699.263714 Exclusion reasons: Q5 [III.ajk]This article's empirical work is concerned only with implementation cost.
120. Boyko B. Bantchev (1998): Putting more meaning in expressions. SIGPLAN Notices 33 (9). Pages 77-83. doi:10.1145/290229.290237 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
121. N. Baradaran, J. Chame, Chun Chen, P. Diniz, M. Hall, Yoon-Ju Lee, Bing Liu & R. Lucas (2003): ECO: an empirical-based compilation and optimization system. In Parallel and Distributed Processing Symposium, 2003. Proceedings. International. doi:10.1109/IPDPS.2003.1213377 Exclusion reasons: Q1–2 [II.ajk]Implementation technique development.
122. J. A. Barnden (1981): Nonsequentiality and Concrete Activity Phases in Discrete-Event Simulation Languages. ACM Transactions on Programming Languages and Systems 3 (3). Pages 293-317. doi:10.1145/357139.357144 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
123. Fred Barnes, Christian Jacobsen & Brian Vinter (2003): RMoX: A Raw-Metal occam Experiment. In Communicating Process Architectures 2003. http://kar.kent.ac.uk/13917/ Exclusion reasons: Q1–2 [III.ajk]This paper is essentially an operating system exposition. While it demonstrates the feasibility of writing an operating system in Occam, this is merely the exploration of the implications of that language design, with no comparative evaluation.
124. M. P. Barnett (1963): Continued operation notation for symbol manipulation and array processing. Communications of the ACM 6 (8). Pages 467-472. doi:10.1145/366707.367587 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
125. M. P. Barnett, J. M. Gerard & A. W. Sambles (1964): Comments on "a continued operation notation". Communications of the ACM 7 (2). Pages 150-152. doi:10.1145/363958.363978 Exclusion reasons: Q5 [III.ajk]This note does not aspire to empiricity.
126. Michael P. Barnett & William M. Ruhsam (1969): SNAP: an experiment in natural language programming. In Proceedings of the May 14-16, 1969, spring joint computer conference. Pages 75-87. doi:10.1145/1476793.1476815 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.
127. Mike Barnett, Manuel Fähndrich, K. Rustan M. Leino, Peter Müller, Wolfram Schulte & Herman Venter (2011): Specification and verification: the Spec# experience. Communications of the ACM 54 (6). Pages 81-91. doi:10.1145/1953122.1953145 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
128. B. M. Barry (1989): Prototyping a real-time embedded system in Smalltalk. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 255-265. doi:10.1145/74877.74904 Exclusion reasons: Q5 [III.ajk]This is an experience report and as such excluded under our protocol.
129. Denis Barthou, Albert Cohen & Jean-François Collard (1998): Maximal static expansion. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 98-106. doi:10.1145/268946.268955 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
130. V.R. Basili & Jr. Reiter, R. W. (1981): A Controlled Experiment Quantitatively Comparing Software Development Approaches. Software Engineering, IEEE Transactions on SE-7 (3). Pages 299-320. doi:10.1109/TSE.1981.230841 Exclusion reasons: Q1–2 [III.ajk]This article reports of an experiment evaluating hypotheses regarding the use of programming methodology. There is no PL design issue here.
131. Victor R. Basili & Barry T. Perricone (1984): Software errors and complexity: an empirical investigation. Communications of the ACM 27 (1). Pages 42-52. doi:10.1145/69605.2085 Exclusion reasons: Q1–2 [II.ajk]No language relevance.
132. David Basin & Grit Denker (2000): Maude versus Haskell: an Experimental Comparison in Security Protocol Analysis. Electronic Notes in Theoretical Computer Science 36. Pages 235-256. doi:10.1016/S1571-0661(05)80141-0 Exclusion reasons: Q5 [III.ajk]This article is analytical in nature.
133. Tania Basso, Regina L. O. Moraes, Bruno P. Sanches & Mario Jino (2009): An Investigation of Java Faults Operators Derived from a Field Data Study on Java Software Faults. Report. http://www.ceset.unicamp.br/~regina/pub/An%20Investigation%20of%20Java%20Faults%20Operators.pdf Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which faults in actual Java programs are described and classified. It does not evaluate language design decisions.
134. Farokh B. Bastani & S. Sitharama Iyengar (1987): The effect of data structures on the logical complexity of programs. Communications of the ACM 30 (3). Pages 250-259. doi:10.1145/214748.214760 Exclusion reasons: Q1–2 [III.ajk]This article reports an experiment, with human participants, intended to determine the effect of data structure choices to program complexity. It has no relevance to programming language design.
135. Daniel Bates, Adam Barth & Collin Jackson (2010): Regular expressions considered harmful in client-side XSS filters. In Proceedings of the 19th international conference on World wide web. New York, NY, USA: ACM. WWW '10. Pages 91-100. doi:10.1145/1772690.1772701 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any programming language design issues.
136. Don Batory, Clay Johnson, Bob MacDonald & Dale von Heeder (2000): Achieving Extensibility through Product-Lines and Domain-Specific Languages: A Case Study. Volume 1844.In Frakes, William (ed.) Software Reuse: Advances in Software Reusability.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 83-153. doi:10.1007/978-3-540-44995-9_8 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a PL design decision.
137. Don Batory, Clay Johnson, Bob MacDonald & Dale von Heeder (2002): Achieving extensibility through product-lines and domain-specific languages: a case study. ACM Transactions on Software Engineering Methodology 11 (2). Pages 191-214. doi:10.1145/505145.505147 Exclusion reasons: Q5 [III.ajk]This article reports on a software development project using certain new technologies which needed to create another new techology. However, it does not evaluate the language design decisions in any meaningful empirical way.
138. F. L. Bauer & H. Wössner (1972): The "Plankalkül" of Konrad Zuse: a forerunner of today's programming languages. Communications of the ACM 15 (7). Pages 678-685. doi:10.1145/361454.361515 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
139. Alan Bawden (2000): First-class macros have types. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 133-141. doi:10.1145/325694.325710 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article does not aspire to empiricity.
140. Till G. Bay, Manuel Oriol & Bertrand Meyer (2012): Release early and often: Developing Software with Origo. Technical Report 581 at Eidgenössische Technische Hochschule Zürich, Department of Computer Science. doi:10.3929/ethz-a-006820313 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
141. Piraye Bayman & Richard E. Mayer (1983): A diagnosis of beginning programmers' misconceptions of BASIC programming statements. Communications of the ACM 26 (9). Pages 677-679. doi:10.1145/358172.358408 Exclusion reasons: Q1–2 [II.ajk]Study of learning; no PL design issue.
142. U. Becker, F. J. Hauck & J. Kleinöder (1998): D2AL-A Design-Based Aspect Language for Distribution Control. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 578. doi:10.1007/3-540-49255-0_125 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.
143. Nels E. Beckman, Duri Kim & Jonathan Aldrich (2011): An Empirical Study of Object Protocols in the Wild. In Proc. ECOOP 2011 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 2-26. doi:10.1007/978-3-642-22655-7_2 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.
144. Andi Bejleri, Andrew Farrell & Patrick Goldsack (2011): Cloudscape: language support to coordinate and control distributed applications in the cloud. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 183-194. doi:10.1145/2095050.2095080 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
145. R. W. Bemer (1959): Automatic programming systems. Communications of the ACM 2 (5). Pages 16. doi:10.1145/368325.1064210 Exclusion reasons: Q1–2 [III.ajk]This is a table, not a report of a study.
146. R. W. Bemer (1959): A checklist of intelligence for programming systems. Communications of the ACM 2 (3). Pages 8-13. doi:10.1145/368300.368320 Exclusion reasons: Q1–2 [III.ajk]This prescriptive article does not report a study.

147. Alan C. Benander & Barbara A. Benander (1989): An empirical study of COBOL programs via a style analyzer: The benefits of good programming style. Journal of Systems and Software 10 (4). Pages 271-279. doi:10.1016/0164-1212(89)90074-5 Exclusion reasons: Q1–2 [II.ajk]Metrics and programming style study. No PL design issue.
148. Alan C. Benander, Barbara A. Benander & Howard Pu (1996): Recursion vs. iteration: An empirical study of comprehension. Journal of Systems and Software 32 (1). Pages 73-82. doi:10.1016/0164-1212(95)00043-7 Exclusion reasons: Q1–2 [III.ajk]This article reports a controlled experiment with human participants comparing the comprehensibility of recursion and iteration in PASCAL. It does not evaluate any language design decisions.
149. Alan Benander, Barbara Benander & Janche Sang (2004): Factors related to the difficulty of learning to program in Java–an empirical study of non-novice programmers. Information and Software Technology 46 (2). Pages 99-107. doi:10.1016/S0950-5849(03)00112-5 Exclusion reasons: Q1–2 [II.ajk]Studies teaching.
150. Izak Benbasat, Albert S. Dexter & Paul S. Masulis (1981): An experimental study of the human/computer interface. Communications of the ACM 24 (11). Pages 752-762. doi:10.1145/358790.358795 Exclusion reasons: Q1–2 [III.ajk]This paper reports a human-subject experiment evaluating different modes of written-command interactive user interfaces. The results, while interesting, appear not to be transferable to programming language design.
151. K. Benkerimi & P. Hill (1992): Object-oriented programming in Gödel: An experiment. Volume 649.In Pettorossi, A. (ed.) Meta-Programming in Logic. Lecture Notes in Computer Science. Pages 177-191. doi:10.1007/3-540-56282-6_12 Exclusion reasons: Q1–2 [II.ajk]No comparison.
152. J. P. Benson & S. H. Saib (1978): A software quality assurance experiment. In Proceedings of the software quality assurance workshop on Functional and performance issues. Pages 87-91. doi:10.1145/800283.811105 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which a specific program was modified to introduce bugs and assertions and then run to determine which bugs were detected by the assertions. This does not evaluate the efficacy of assertions from the point of view of a programmer, merely whether they can be used to detect bugs.
153. Nick Benton, Luca Cardelli & Cédric Fournet (2002): Modern Concurrency Abstractions for C#. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 415-440. doi:10.1007/3-540-47993-7_18 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
154. David Bergantz & Johnette Hassell (1991): Information relationships in PROLOG programs: how do programmers comprehend functionality?. International Journal of Man-Machine Studies 35 (3). Pages 313-328. doi:10.1016/S0020-7373(05)80131-2 Exclusion reasons: Q1–2 [II.ajk]This article does not seem to evaluate language design decisions.
155. Alexandre Bergel, Stéphane Ducasse & Oscar Nierstrasz (2005): Classbox/J: controlling the scope of change in Java. In OOPSLA ’05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 177-189. doi:10.1145/1094811.1094826 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article does not aspire to empiricity (even the "case study" is analytical, as it explores the implications of the construct).
156. L. Berger, A. M. Dery & M. Fornarino (1998): Interactions between Objects: An Aspect of Object-Oriented Languages. In ECOOP’98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 586. doi:10.1007/3-540-49255-0_126 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.
157. A. Michael Berman (1994): Does Scheme enhance an introductory programming course?: some preliminary empirical results. SIGPLAN Notices 29 (2). Pages 44-48. doi:10.1145/181748.181758 Exclusion reasons: Q1–2 [III.ajk]This article reports on a study in which a programming course was changed from BASIC to Scheme, and its effect on self-reported student satisfaction was monitored. This does not in any meaningful way evaluate the efficacy of the difference between the two languages.
158. Gerald M. Berns (1984): Assessing software maintainability. Communications of the ACM 27 (1). Pages 14-23. doi:10.1145/69605.357965 Exclusion reasons: Q5 [III.ajk]This article defines and uses a code metric. It does not evaluate a language design decision.
159. Arthur Bernstein (1980): Output Guards and Nondeterminism in "Communicating Sequential Processes". ACM Transactions on Programming Languages and Systems 2 (2). Pages 234-238. doi:10.1145/357094.357101 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
160. R. E. Berry & B. A.E. Meekings (1985): A style analysis of C programs. Communications of the ACM 28 (1). Pages 80-88. doi:10.1145/2465.2469 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.
161. Gerard Berry & Gerard Boudol (1990): The chemical abstract machine. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 81-94. doi:10.1145/96709.96717 Exclusion reasons: Q1–2 [II.ajk]Formal theory development.
162. G. Berry, S. Ramesh & R. K. Shyamasundar (1993): Communicating reactive processes. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 85-98. doi:10.1145/158511.158526 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
163. Adam Betts, Nathan Chong, Alastair Donaldson, Shaz Qadeer & Paul Thomson (2012): GPUVerify: a verifier for GPU kernels. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 113-132. doi:10.1145/2384616.2384625 Exclusion reasons: Q1–2 [III.ajk]This article does not have language design relevance.
164. Antoine Beugnard (2006): Method overloading and overriding cause encapsulation flaw: an experiment on assembly of heterogeneous components. In Proceedings of the 2006 ACM symposium on Applied computing. Pages 1424-1428. doi:10.1145/1141277.1141608 Exclusion reasons: Q5 [III.ajk]Despite the use of the term "experiment", this article is analytical in nature and does not report an empirical study.
165. Jean Bezivin (1987): Some experiments in object-oriented simulation. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 394-405. doi:10.1145/38765.38843 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions (no comparison).
166. Karthikeyan Bhargavan, Cédric Fournet & Andrew D. Gordon (2008): Verifying policy-based web services security. ACM Transactions on Programming Languages and Systems 30 (6). doi:10.1145/1391956.1391957 Exclusion reasons: Q1–2 [II.ajk]No comparison.
167. Pamela Bhattacharya & Iulian Neamtiu (2011): Assessing programming language impact on development and maintenance: a study on c and c++. In Proceedings of the 33rd International Conference on Software Engineering. New York, NY, USA: ACM. ICSE ’11. Pages 171-180. doi:10.1145/1985793.1985817 Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study comparing C and C++ for various measures of efficacy. However, there is no clear language design decision at play, since the languages are similar in many ways and different in many others, and there is no clear answer to the question what difference is it that explains the measured efficacy differences.
168. Marina Biberstein, Joseph (Yossi) Gil & Sara Porat (2001): Sealing, Encapsulation, and Mutablility. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 28-52. doi:10.1007/3-540-45337-7_3 Exclusion reasons: Q1–2 [II.ajk]Program analysis study.
169. Lubomir Bic & Craig Lee (1987): A data-driven model for a subset of logic programming. ACM Transactions on Programming Languages and Systems 9 (4). Pages 618-645. doi:10.1145/29873.31333 Exclusion reasons: Q1–2 [II.ajk]According to the abstract, this develops an implementation technique.
170. Michael Allen Bickel (1987): Automatic correction to misspelled names: a fourth-generation language approach. Communications of the ACM 30 (3). Pages 224-228. doi:10.1145/214748.214756 Exclusion reasons: Q1–2 [II.ajk]Based on the abstract, this is an exposition of a new technique, with no evaluation, and relevance to programming languages is doubtful.
171. Kevin Bierhoff, Nels E. Beckman & Jonathan Aldrich (2009): Practical API Protocol Checking with Access Permissions. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 195-219. doi:10.1007/978-3-642-03013-0_10 Exclusion reasons: Q5 [III.ajk]This article reports on a series of "case studies" (as the article calls them) in which the authors have used their annotation language to annotate existing software and their corresponding static analysis tool to statically analyze the annotated software. The investigations are analytical in nature, and do not aspire to empiricality.
172. Gavin Bierman, Erik Meijer & Wolfram Schulte (2005): The Essence of Data Access in Cω: The Power is in the Dot!. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 287-311. doi:10.1007/11531142_13 Exclusion reasons: Q1–2 [II.ajk]Formal exposition of a language.
173. Gavin Bierman & Alisdair Wren (2005): First-Class Relationships in an Object-Oriented Language. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 262-286. doi:10.1007/11531142_12 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
174. Gavin Bierman, Matthew Parkinson & James Noble (2008): UpgradeJ: Incremental Typechecking for Class Upgrades. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 235-259. doi:10.1007/978-3-540-70592-5_11 Exclusion reasons: Q5 [III.ajk]This analytical presentation of a new construction does not have empirical aspirations.
175. Gavin Bierman, Erik Meijer & Mads Torgersen (2010): Adding Dynamic Types to C#. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 76-100. doi:10.1007/978-3-642-14107-2_5 Exclusion reasons: Q5 [II.ajk]Type-theoretical development.
176. Ted Biggerstaff (1998): A perspective of generative reuse. Annals of Software Engineering 5 (1). Pages 169-226. doi:10.1023/A:1018924407841 Exclusion reasons: Q6 Q7 [III.ajk]This article is essentially a review and a position statement, advocating and evaluating the use of code-generation techniques for

handling reuse. There is one point where empirical evidence is discussed, but it's presented as motivation, not as bearing on the article's main point.

177. Brian Billard (1981): Polynomial manipulation with APL. Communications of the ACM 24 (7). Pages 457-465. doi:10.1145/358699.358716 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

178. Tim Bingham, Nancy Hobbs & Dave Husson (1993): Experiences developing and using an object-oriented library for program manipulation. In OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. Pages 83-89. doi: 10.1145/165854.165872 Exclusion reasons: Q5 [III.ajk]This article reports on an application developed in C++. The article makes a claims as to the efficacy of the use of object-orientation in lieu of traditional programming techniques, but those claims are, in the article, bare assertions with no support.

179. Davey Binkley, Marcia Davis, Dawn Lawrie & Christopher Morrell (2009): To CamelCase or Under_score. In Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on. Pages 158-167. doi:10.1109/ICPC.2009.5090039 Exclusion reasons: Q1–2 [II.ajk]This article studies naming conventions which are usually not regarded as language design issues.

180. Stefano Bistarelli & Francesca Rossi (2001): Semiring-based constraint logic programming: syntax and semantics. ACM Transactions on Programming Languages and Systems 23 (1). Pages 1-29. doi:10.1145/383721.383725 Exclusion reasons: Q5 [II.ajk]Theoretical work.

181. Sandip K. Biswas (1995): Higher-order functors with transparent signatures. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 154-163. doi:10.1145/199448.199478 Exclusion reasons: Q5 [II.ajk]Formal theoretical study.

182. M.Z. Bjelica, B. Mrazovac & N. Teslic (2011): Evaluation of the available scripting languages for home automation networks: Real world case study. Volume 2.In Telecommunication in Modern Satellite Cable and Broadcasting Services (TELSIKS), 2011 10th International Conference on. Pages 611-614. doi:10.1109/TELSKS.2011.6143187 Exclusion reasons: Q5 [III.ajk]The test scripts in this study are prewritten presumably by the investigators themselves. This seems to make the efficacy aspect of this work analytic – what can these languages do instead of what these languages actually do in the hands of the intended programmers.

183. Andrew Black, Norman Hutchinson, Eric Jul & Henry Levy (1986): Object structure in the Emerald system. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 78-86. doi:10.1145/28697.28706 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

184. Andrew P. Black & Mark P. Immel (1993): Encapsulating Plurality. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 57-79. doi:10.1007/3-540-47910-4_5 Exclusion reasons: Q1–2 [II.ajk]Language feature exposition.

185. Alan Blackwell & Rob Hague (2001): Designing a Programming Language for Home Automation. In PPIG 2001. (Found in http://ppig.org/workshops/13th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article discusses design of two specific languages with no apparent evalative intent.

186. Alan F. Blackwell (2003): Cognitive Dimensions of tangible programming techniques. In PPIG 2003. (Found in http://ppig.org/workshops/15th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not discuss matters relating to programming language design.

187. Alan Blackwell & Nick Collins (2005): The Programming Language as a Musical Instrument. In PPIG 2005. (Found in http://ppig.org/workshops/17th-programme.html.) Exclusion reasons: Q5 [III.ajk]This article does not report an evaluative empirical study.

188. Alan F. Blackwell (2006): Psychological Issues in End-User Programming. Volume 9.In Lieberman, Henry and Paternò, Fabio and Wulf, Volker (ed.) End User Development.Springer Netherlands. Human–Computer Interaction Series. Pages 9-30. doi:10.1007/1-4020-5386-X_2 Exclusion reasons: Q7 [III.ajk]This article provides an overview of research but does not discuss the empirical evidence involved.

189. Alan F. Blackwell, Jennifer A. Rode & Eleanor F. Toye (2009): How do we program the home? Gender, attention investment, and the psychology of programming at home. International Journal of Human-Computer Studies 67 (4). Pages 324-341. doi:10.1016/j.ijhcs.2008.09.011 Exclusion reasons: Q1–2 [III.ajk]This article does not discuss programming languages.

190. Edwin Blake & Steve Cook (1987): On Including Part Hierarchies in Object-Oriented Languages, with an Implementation in Smalltalk. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 41-50. doi:10.1007/3-540-47891-4_5 Exclusion reasons: Q5 [III.ajk]This is an analytical study with no aspirations to empiricity (despite using the word "experiment" to describe exploration of the implications of the concept at hand).

191. John M. Blatt (1960): Comments from a FORTRAN user. Communications of the ACM 3 (9). Pages 501-505. doi:10.1145/367390.367404 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

192. Martin Blom (2006): Empirical Evaluations of Semantic Aspects in Software Development. Karlstad University, Faculty of Economic Sciences, Communication and IT. Karlstad University Studies 2006:26. http://kau.diva-portal.org/smash/record.jsf?pid=diva2:6529 Exclusion reasons: Q1–2 [III.ajk]This thesis does not evaluate any language design decisions – it focuses mostly on development methodology.

193. Bard Bloom, John Field, Nathaniel Nystrom, Johan Östlund, Gregor Richards, Rok Strniša, Jan Vitek & Tobias Wrigstad (2009): Thorn: robust, concurrent, extensible scripting on the JVM. In Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. OOPSLA '09. Pages 117-136. doi:10.1145/1640089.1640098 Exclusion reasons: Q5 [II.ajk]The abstract does not divulge any empirical evaluation.

194. Edward K. Blum (1988): The semantics and complexity of parallel programs for vector computations. Part I: A case study using ADA. BIT Numerical Mathematics 28 (3). Pages 530-551. (10.1007/BF01941132.) doi:10.1007/BF01941132 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.

195. A. Bobkowska (2003): Cognitive Dimensions questionnaire applied to visual languages evaluation – a case study. In PPIG 2003. (Found in http://ppig.org/workshops/15th-programme.html.) Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded from our definition of programming languages.

196. Daniel G. Bobrow & Bertram Raphael (1964): A comparison of list-processing computer languages: including a detailed comparison of COMIT, IPL-V, LISP 1.5, and SLIP. Communications of the ACM 7 (4). Pages 231-240. doi:10.1145/364005.364057 Exclusion reasons: Q5 [III.ajk]This article presents an analytical comparison of several languages.

197. Daniel G. Bobrow (1980): Managing Reentrant Structures Using Reference Counts. ACM Transactions on Programming Languages and Systems 2 (3). Pages 269-273. doi:10.1145/357103.357104 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

198. Daniel G. Bobrow, Kenneth Kahn, Gregor Kiczales, Larry Masinter, Mark Stefik & Frank Zdybel (1986): CommonLoops: merging Lisp and object-oriented programming. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 17-29. doi:10.1145/28697.28700 Exclusion reasons: Q1–2 [II.ajk]Language exposition

199. Robert L. Bocchino & Vikram S. Adve (2011): Types, Regions, and Effects for Safe Programming with Object-Oriented Parallel Frameworks. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 306-332. doi: 10.1007/978-3-642-22655-7_15 Exclusion reasons: Q5 [III.ajk]The evaluation in this article is purely analytical, with no aspiration to empiricity.

200. G. V. Bochmann (1973): Multiple exits from a loop without the GOTO. Communications of the ACM 16 (7). Pages 443-444. doi:10.1145/362280.362300 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.

201. S{\'e}bastien Bocq & Koen Daenen (2012): Molecule: using monadic and streaming I/O to compose process networks on the JVM. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 315-334. doi:10.1145/2384616.2384640 Exclusion reasons: Q1–2 [III.ajk]This article uses only performance measures as evaluation.

202. Eric Bodden, Laurie Hendren & Ondřej Lhoták (2007): A Staged Static Program Analysis to Improve the Performance of Runtime Monitoring. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609. Pages 525-549. doi:10.1007/978-3-540-73589-2_25 Exclusion reasons: Q1–2 [III.ajk]This article reports a study of a static analysis; there is no PL design issue here.

203. Eric Bodden, Patrick Lam & Laurie Hendren (2012): Partially Evaluating Finite-State Runtime Monitors Ahead of Time. ACM Transactions on Programming Languages and Systems 34 (2). Pages 7:1–7:52. doi:10.1145/2220365.2220366 Exclusion reasons: Q1–2 [III.ajk]It is questionable whether the constructs in this work amount to language design options; in any case, the evaluations are not comparative and thus there is no design decision efficacy issue here.

204. Rastislav Bodik, Satish Chandra, Joel Galenson, Doug Kimelman, Nicholas Tung, Shaon Barman & Casey Rodarmor (2010): Programming with angelic nondeterminism. In Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 339-352. doi:10.1145/1706299.1706339 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

205. B. W. Boehm (1981): An Experiment in Small-Scale Application Software Engineering. Software Engineering, IEEE Transactions on SE-7 (5). Pages 482-493. doi:10.1109/TSE.1981.231110 Exclusion reasons: Q1–2 [III.ajk]This empirical study of software development process does not present a PL design issue.

206. B.W. Boehm (1981): Developing small-scale application software products: Some experimental results.. SOFTWARE WORLD 12 (1). Pages 2-

8. http://md1.csa.com/partners/viewrecord.php?requester=gs&collection=TRD&recid=0395525CI&q=intitle%3Aexperimental+%22programming+language%22&uid=788456873&setcookie=yes Exclusion reasons: Q1–2 [III.ajk]This article appears to be an earlier version of boehm-1981, containing largely the same verbatim text. There does not appear to be any reference from one to the other, though. Decision copied from that article.

207. Hans-J. Boehm, Robert Cartwright, Mark Riggle & Michael J. O'Donnell (1986): Exact real arithmetic: a case study in higher order programming. In Proceedings of the 1986 ACM conference on LISP and functional programming. New York, NY, USA: ACM. Pages 162-173. doi:10.1145/319838.319860 Exclusion reasons: Q5 [III.ajk]This article has very little programming language design relevance; what relevance it has (the feasibility of an exact real number type) is not empirical, rather it is analytical-constructive.

208. Hans-J. Boehm (2003): Destructors, finalizers, and synchronization. In Proc. 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 262-272. doi:10.1145/604131.604153 Exclusion reasons: Q5 [III.ajk]This article is analytical, not empirical.

209. Deborah A. Boehm-Davis, Sylvia B. Sheppard & John W. Bailey (1982): An empirical evaluation of language-tailored PDLs. Human Factors and Ergonomics Society Annual Meeting Proceedings 26 (11). Pages 984-988. doi:10.1177/154193128202601117 http://pro.sagepub.com/content/26/11/984.abstract Exclusion reasons: Q1–2 [II.ajk]Studies program design languages, not programming languages.

210. Deborah A. Boehm-Davis, Sylvia B. Sheppard & John W. Bailey (1987): Program design languages: How much detail should they include?. International Journal of Man-Machine Studies 27 (4). Pages 337-347. doi:10.1016/S0020-7373(87)80002-0 Exclusion reasons: Q1–2 [II.ajk]Design languages are not, by definition, programming languages.

211. Deborah A. Boehm-Davis & Lyle S. Ross (1992): Program design methodologies and the software development process. International Journal of Man-Machine Studies 36 (1). Pages 1-19. doi:10.1016/0020-7373(92)90050-U Exclusion reasons: Q1–2 [II.ajk]No language design issue.

212. Deborah A. Boehm-Davis, Robert W. Holt & Alan C. Schultz (1992): The role of program structure in software maintenance. International Journal of Man-Machine Studies 36 (1). Pages 21-63. doi:10.1016/0020-7373(92)90051-L Exclusion reasons: Q1–2 [III.ajk]Evaluates programming styles, not language design decisions.

213. C. Bogart, M. Burnett, A. Cypher & C. Scaffidi (2008): End-user programming in the wild: A field study of CoScripter scripts. In Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on. Pages 39-46. doi:10.1109/VLHCC.2008.4639056 Exclusion reasons: Q1–2 [III.ajk]This article presents a study in which the actual use of a particular lagnuage is examined. It does not evaluate any design decisions.

214. Aaron Bohannon, J. Nathan Foster, Benjamin C. Pierce, Alexandre Pilkiewicz & Alan Schmitt (2008): Boomerang: resourceful lenses for string data. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 307-419. doi:10.1145/1328438.1328487 Exclusion reasons: Q5 [III.ajk]This article presents and analyzes a new construction. It also briefly discusses the authors' experience. There were no empirical aspirations.

215. Mikolaj Bojanczyk, Laurent Braud, Bartek Klin & Slawomir Lasota (2012): Towards nominal computation. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 401-412. doi:10.1145/2103656.2103704 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

216. Boris Bokowski & Markus Dahm (1998): Poor Man's Genericity for Java. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 587. doi:10.1007/3-540-49255-0_182 Exclusion reasons: Q5 [III.ajk]This study does not aspire to empiricity.

217. Jeffrey Bonar & Elliot Soloway (1983): Uncovering principles of novice programming. In Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 10-13. doi:10.1145/567067.567069 Exclusion reasons: Q1–2 [III.ajk]This article describes the results of a qualitative study trying to understand the sources of novice bafflement in programming. It has little direct relevance to language design evaluation.

218. Viviana Bono, Jarek Kuśmierek & Mauro Mulatero (2012): Magda: A New Language for Modularity. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 560-588. doi:10.1007/978-3-642-31057-7_25 Exclusion reasons: Q1–2 [III.ajk]This article does not aspire to empiricity.

219. Grady Booch & Michael Vilot (1990): The design of the C++ Booch Components. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 1-11. doi:10.1145/97945.97947 Exclusion reasons: Q1–2 [II.ajk]No comparison language.

220. Simon P Booth & Simon B Jones (1996): Are Ours Really Smaller Than Theirs?. In Proc. 1996 Glasgow Workshop on Functional Programming. http://ftp.dcs.glasgow.ac.uk/fp/workshops/fpw96/Booth.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

221. E. Borger & W. Schulte (2000): A practical method for specification and analysis of exception handling-a Java/JVM case study. Software Engineering, IEEE Transactions on 26 (9). Pages 872 -887. doi:10.1109/32.877847 Exclusion reasons: Q5 [II.ajk]Formal development

222. I. Borne & S. Despres (1993): A Weighted Pattern Matching to Help Smalltalk Class Creation . In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

223. Alan Borning (1981): The Programming Language Aspects of ThingLab, a Constraint-Oriented Simulation Laboratory. ACM Transactions on Programming Languages and Systems 3 (4). Pages 353-387. doi:10.1145/357146.357147 Exclusion reasons: Q1–2 [III.ajk]This article is a system exposition.

224. Alan H. Borning & Daniel H. H. Ingalls (1982): A type declaration and inference system for smalltalk. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 133-141. doi:10.1145/582153.582168 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

225. Alan Borning & Tim O'Shea (1987): Deltatalk: An Empirically and Aesthetically Motivated Simplification of the Smalltalk-80 Language. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276 . Pages 1-10. doi:10.1007/3-540-47891-4_1 Exclusion reasons: Q3 [III.ajk]This article provides guidelines for language design based on empirical studies reported at reference 15. Reference 15 (Tim O'Shea's position paper to the panel "The Learnability of Object-oriented Programming Systems", at page 502 of OOPSLA 1986 proceedings, not available online) is a short summary of the studies and lacks sufficient detail to evaluate their quality. A more detailed report has not been identified.

226. Jan van den Bos, R. Plasmeijer & Jan W. M. Stroet (1981): Process Communication Based on Input Specifications. ACM Transactions on Programming Languages and Systems 3 (3). Pages 224-250. doi:10.1145/357139.357141 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

227. Jan Van Den Bos (1988): Abstract interaction tools: a language for user interface management systems. ACM Transactions on Programming Languages and Systems 10 (2). Pages 215-247. doi:10.1145/42190.42191 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.

228. Edwin Bos (1996): The use and acquisition of artificial language: Some insights from psycholinguistics. Computers in Human Behavior 12 (3). Pages 425-447. doi:10.1016/0747-5632(96)00017-9 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any programming language design decisions.

229. A. Bossi, M. Bugliesi, M. Gabbrielli, G. Levi & M. C. Meo (1993): Differential logic programming. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 359-370. doi:10.1145/158511.158689 Exclusion reasons: Q1–2 [II.ajk]Formal semantics development.

230. Raymond Boute (2005): Functional declarative language design and predicate calculus: a practical approach. ACM Transactions on Programming Languages and Systems 27 (5). Pages 988-1047. doi:10.1145/1086642.1086647 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

231. A. Bowles, D. Robertson, W. Vasconcelos, M. Vargas-Vera & D. Bental (1994): Applying Prolog programming techniques. International Journal of Human-Computer Studies 41 (3). Pages 329-350. doi:10.1006/ijhc.1994.1062 Exclusion reasons: Q1–2 [II.ajk]This article appears not to evaluate any language design decisions.

232. Chandrasekhar Boyapati, Barbara Liskov & Liuba Shrira (2003): Ownership types for object encapsulation. In Proc. 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 213-223. doi:10.1145/604131.604156 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

233. Raymond F. Boyce, Donald D. Chamberlin, III W. Frank King & Michael M. Hammer (1975): Specifying queries as relational expressions: the SQUARE data sublanguage. Communications of the ACM 18 (11). Pages 621-628. doi:10.1145/361219.361221 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

234. John Boyland, James Noble & William Retert (2001): Capabilities for Sharing: A Generalisation of Uniqueness and Read-Only. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 2-27. doi:10.1007/3-540-45337-7_2 Exclusion reasons: Q5 [II.ajk]Formal development.

235. John Tang Boyland & William Retert (2005): Connecting effects and uniqueness with adoption. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 283-295. doi:10.1145/1040305.1040329 Exclusion reasons: Q1–2 [II.ajk]Formal type-theoretic development.

236. Gilad Bracha, Martin Odersky, David Stoutamire & Philip Wadler (1998): Making the future safe for the past: adding genericity to the Java programming language. In Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. New York, NY, USA: ACM. OOPSLA '98. Pages 183–200. doi:10.1145/286936.286957 Exclusion reasons: Q5 [II.ajk]This article does not aspire to empiricity.

237. Gilad Bracha, Peter von der Ahé, Vassili Bykov, Yaron Kashai, William Maddox & Eliot Miranda (2010): Modules as Objects in Newspeak. In

Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 405-428. doi: 10.1007/978-3-642-14107-2_20 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

238. Walt Brainerd (1978): Fortran 77. Communications of the ACM 21 (10). Pages 806-820. doi:10.1145/359619.359621 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

239. Søren Brandt & Jørgen Lindskov Knudsen (1996): Generalising the BETA type system. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 421-448. doi:10.1007/BFb0053072 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

240. P. Branquart, J. Lewi, M. Sintzoff & P. L. Wodon (1971): The composition of semantics in Algol 68. Communications of the ACM 14 (11). Pages 697-708. doi:10.1145/362854.362874 Exclusion reasons: Q1–2 [II.ajk]Language exposition

241. John Brant, Brian Foote, Ralph E. Johnson & Donald Roberts (1998): Wrappers to the rescue. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 396-417. doi:10.1007/BFb0054101 Exclusion reasons: Q1–2 [II.ajk]No comparison language.

242. Harvy Bratman, Julien Green, John Stockman & Albert R. Watson (1959): Recommendations of the SHARE ALGOL Committee. Communications of the ACM 2 (10). Pages 24-26. doi:10.1145/368453.1127875 http://dl.acm.org/citation.cfm?id=1127875 Exclusion reasons: Q1–2 [III.ajk]This language design note does not evaluate its recommendations.

243. Martin Bravenboer & Eelco Visser (2004): Concrete syntax for objects: domain-specific language embedding and assimilation without restrictions. In OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 365-383. doi:10.1145/1028976.1029007 Exclusion reasons: Q5 [III.ajk]This constructive-analytical paper has no empirical aspirations.

244. Jean-Pierre Briot & Akinori Yonezawa (1987): Inheritance and Synchronization in Concurrent OOP. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276 . Pages 32-40. doi:10.1007/3-540-47891-4_4 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

245. J.-P. Briot & P. Cointe (1989): Programming with explicit metaclasses in Smalltalk-80. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 419-431. doi:10.1145/74877.74921 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

246. Jean-Pierre Briot (1989): Actalk: A Testbed for Classifying and Designing Actor Languages in the Smalltalk-80 Environment. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 109-129. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890109.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

247. F. Brito e Abreu & W. Melo (1996): Evaluating the impact of object-oriented design on software quality . In Software Metrics Symposium, 1996., Proceedings of the 3rd International. Pages 90-99. doi:10.1109/METRIC.1996.492446 Exclusion reasons: Q1–2 [III.ajk]If I understand this article correctly, all the groups used OO. Thus, there was no real efficacy evaluation of OO as a design decision, nor do I see any other design decisions at play.

248. W. R. Brittenham, K. Clark, G. Kuss & H. Thompson (1959): SALE, a simple algebraic language for engineers. Communications of the ACM 2 (10). Pages 22-24. doi:10.1145/368453.368464 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity.

249. Niklas Broberg & David Sands (2010): Paralocks: role-based information flow control and beyond. In Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 431-444. doi:10.1145/1706299.1706349 Exclusion reasons: Q5 [II.ajk]Theoretical work.

250. Antonio Brogi & Paolo Ciancarini (1991): The concurrent language, Shared Prolog. ACM Transactions on Programming Languages and Systems 13 (1). Pages 99-123. doi:10.1145/114005.102807 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity.

251. Antonio Brogi, Paolo Mancarella, Dino Pedreschi & Franco Turini (1994): Modular logic programming. ACM Transactions on Programming Languages and Systems 16 (4). Pages 1361-1398. doi:10.1145/183432.183528 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

252. Ruven E. Brooks (1980): Studying programmer behavior experimentally: the problems of proper methodology. Communications of the ACM 23 (4). Pages 207-213. doi:10.1145/358841.358847 Exclusion reasons: Q1–2 [III.ajk]This article discusses methodological issues in programmer behaviour experimentation. As such, it is out of scope of this mapping study. However, its reference list appears very useful for snowballing.

253. Ruven Brooks (1983): Towards a theory of the comprehension of computer programs. International Journal of Man-Machine Studies 18 (6). Pages 543-554. doi:10.1016/S0020-7373(83)80031-5 Exclusion reasons: Q1–2 [II.ajk]No language design decision evaluation.

254. Ruven Brooks (1990): Categories of programming knowlege and their application. International Journal of Man-Machine Studies 33 (3). Pages 241-246. doi:10.1016/S0020-7373(05)80118-X Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.

255. A. Brooks, J. Miller, M. Roper & Wood M. (1992): Criticisms of an Empirical Study of Recursion and Iteration. Technical report EFoCS-1-92 at Department of Computer Science, University of Strathclyde.. http://staff.unak.is/andy/ResearchPapers/EFoCS-1-92.pdf Exclusion reasons: Q1–2 [III.ajk]This article is a critique of sinha-1992 and does not report a study.

256. RUVEN BROOKS (1999): Towards a theory of the cognitive processes in computer programming. International Journal of Human-Computer Studies 51 (2). Pages 197-211. doi:10.1006/ijhc.1977.0306 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

257. Gerald Brose (1998): Towards an Access Control Policy Language for CORBA. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 587. doi:10.1007/3-540-49255-0_60 Exclusion reasons: Q5 [III.ajk]This one-page abstract does not describe empirical research.

258. S. A. Brown, C. E. Drayton & B. Mittman (1963): A description of the APT language. Communications of the ACM 6 (11). Pages 649-658. doi: 10.1145/368310.368322 Exclusion reasons: Q1–2 [II.ajk]Language description only, based on the abstract.

259. P. J. Brown (1967): The ML/I macro processor. Communications of the ACM 10 (10). Pages 618-623. doi:10.1145/363717.363746 Exclusion reasons: Q1–2 [III.ajk]This language exposition does not report a study.

260. P. J. Brown (1972): Levels of language for portable software. Communications of the ACM 15 (12). Pages 1059-1062. doi:10.1145/361598.361624 Exclusion reasons: Q1–2 [II.ajk]Implementation technique study.

261. T. Brown & M. Klerer (1975): The effect of language design on time-sharing operational efficiency. International Journal of Man-Machine Studies 7 (2). Pages 233-247. doi:10.1016/S0020-7373(75)80008-3 Exclusion reasons: Q5 [II.ajk]No empirical evaluation; doubtful that there is an actual language design issue.

262. Peter Brown (1984): Languages: three interviews. Communications of the ACM 27 (4). Pages 352-355. doi:10.1145/358027.358046 Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.

263. Manfred Broy & Greg Nelson (1994): Adding fair choice to Dijkstra's calculus. ACM Transactions on Programming Languages and Systems 16 (3). Pages 924-938. doi:10.1145/177492.177727 Exclusion reasons: Q1–2 [II.ajk]Theoretical development of a specification language.

264. Kim Bruce & John C. Mitchell (1992): PER models of subtyping, recursive types and higher-order polymorphism. In Proc. 19th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 316-327. doi:10.1145/143165.143230 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

265. Kim B. Bruce (1993): Safe type checking in a statically-typed object-oriented programming language. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 285-298. doi:10.1145/158511.158650 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

266. Kim B. Bruce, Angela Schuett & Robert van Gent (1995): PolyTOIL: A Type-Safe Polymorphic Object-Oriented Language. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 27-51. doi:10.1007/3-540-49538-X_3 Exclusion reasons: Q5 [II.ajk]Formal exposition of a single language.

267. Kim B. Bruce, Leaf Petersen & Adrian Fiech (1997): Subtyping is not a good "match" for object-oriented languages. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 104-127. doi:10.1007/BFb0053376 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

268. Kim B. Bruce, Martin Odersky & Philip Wadler (1998): A statically safe alternative to virtual types. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 523-549. doi:10.1007/BFb0054106 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

269. Kim B. Bruce, Angela Schuett, Robert van Gent & Adrian Fiech (2003): PolyTOIL: A type-safe polymorphic object-oriented language. ACM Transactions on Programming Languages and Systems 25 (2). Pages 225-290. doi:10.1145/641888.641891 Exclusion reasons: Q5 [II.ajk]Formal theoretical basis, no empirics.

270. Kim B. Bruce & J. Nathan Foster (2004): LOOJ: Weaving LOOM into Java. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 390-414. doi:10.1007/978-3-540-24851-4_18 Exclusion reasons: Q5 [III.ajk]This article discusses a new construct in theoretical terms, with no aspiration for empiricity.

271. Achim D. Brucker & Burkhart Wolff (2008): Extensible Universes for Object-Oriented Data Models. In Proc. ECOOP 2008 European Conference on

Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 438-462. doi:10.1007/978-3-540-70592-5_19 Exclusion reasons: Q5 [III.ajk]This study has no aspiration for empiricity.

272. Ciarán Bryce, Chrislain Razafimahefa & Michel Pawlak (2002): Lana: An Approach to Programming Autonomous Systems. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 281-308. doi:10.1007/3-540-47993-7_13 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
273. D. Budgen & A. Pohthong (1999): Component reuse in software design: an observational study. In Software Technology and Engineering Practice, 1999. STEP '99. Proceedings. Pages 63-72. doi:10.1109/STEP.1999.798480 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design issues.
274. Francisco Bueno, María García de la Banda & Manuel Hermenegildo (1999): Effectivness of abstract interpretation in automatic parallelization: a case study in logic programming. ACM Transactions on Programming Languages and Systems 21 (2). Pages 189-239. doi:10.1145/316686.316688 Exclusion reasons: Q1–2 [III.ajk]This article evaluates techniques for automatic improvement of programs, which is not a language design matter.
275. Michele Bugliesi & Giuseppe Castagna (2001): Secure safe ambients. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 222-235. doi:10.1145/360204.360223 Exclusion reasons: Q1–2 [II.ajk]Type-theoretic work.
276. P. A. Buhr & C. R. Zarnke (1988): Nesting in an Object Oriented Language is NOT for the Birds. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 128-145. doi:10.1007/3-540-45910-3_8 Exclusion reasons: Q5 [III.ajk]This is an analytical study with no aspiration to empiricality.
277. Peter A. Buhr & Ashif S. Harji (2005): Implicit-signal monitors. ACM Transactions on Programming Languages and Systems 27 (6). Pages 1270-1343. doi:10.1145/1108970.1108975 Exclusion reasons: Q5 [III.ajk]This article presents extensive historical and analytic arguments of the efficacy of providing implicit-signal monitors. It also contains an arguably empirical performance comparison of three ways to actually simulate implict signaling using explicit signaling. That comparison has no bearing on the efficacy question, as far as I can tell; it is concerned more on whether implicit signaling is useful even without language support.
278. Sebastian Burckhardt, Alexandro Baldassin & Daan Leijen (2010): Concurrent programming with revisions and isolation types. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications. Pages 691-707. doi: 10.1145/1869459.1869515 Exclusion reasons: Q5 [III.ajk]This article's evaluative work is analytical in nature.
279. Sebastian Burckhardt, Manuel Fähndrich, Daan Leijen & Benjamin Wood (2012): Cloud Types for Eventual Consistency. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 283-307. doi:10.1007/978-3-642-31057-7_14 Exclusion reasons: Q5 [II.ajk]No empirical evaluation.
280. Kenneth Burkhardt (1976): EMPP: An extensible multiprogramming system for experimental psychology. Behavior Research Methods 8 (2). Pages 239-244. doi:10.3758/BF03201784 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
281. Rachel Burrows, Fabiano C. Ferrari, Otávio A. L. Lemos, Alessandro Garcia & François. Taïani (2010): The Impact of Coupling on the Fault-Proneness of Aspect-Oriented Programs: An Empirical Study. In Software Reliability Engineering (ISSRE), 2010 IEEE 21st International Symposium on. Pages 329-338. doi:10.1109/ISSRE.2010.33 Exclusion reasons: Q1–2 [II.ajk]Metrics study; no PL design issue.
282. R. M. Burstall, D. B. MacQueen & D. T. Sannella (1980): HOPE: An experimental applicative language. In Proceedings of the 1980 ACM conference on LISP and functional programming. Pages 136-143. doi:10.1145/800087.802799 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
283. F. Warren Burton (1990): Type extension through polymorphism. ACM Transactions on Programming Languages and Systems 12 (1). Pages 135-138. doi:10.1145/77606.214515 Exclusion reasons: Q5 [III.ajk]This brief paper does not aspire to empiricity.
284. Raymond P.L. Buse, Caitlin Sadowski & Westley Weimer (2011): Benefits and barriers of user evaluation in software engineering research. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 643-656. doi:10.1145/2048066.2048117 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate nor summarise or consolidate papers evaluating language design decisions.
285. Pauli Byckling (2004): Roles of variables and strategic programming knowledge. In PPIG 2004. (Found in http://ppig.org/workshops/16th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article summarizes ongoing research on teaching of programming. There is no PL design issue present.
286. Pauli Byckling, Petri Gerdt & Jorma Sajaniemi (2005): Roles of variables in object-oriented programming. In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 350-355. doi:10.1145/1094855.1094972 Exclusion reasons: Q1–2 [III.ajk]This article studies a conceptual teaching aid and has no PL design relevance.
287. A. P. W. Böhm & R. R. Oldehoeft (1994): Two issues in parallel language design. ACM Transactions on Programming Languages and Systems 16 (6). Pages 1675-1683. doi:10.1145/197320.197325 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
288. Kai Böllert (1998): Aspect-oriented programming case study: System management application. In Aspect-Oriented Programming Workshop 1998. (Position paper for Aspect-Oriented Programming Workshop 1998.) http://trese.cs.utwente.nl/aop-ecoop98/papers/Boellert.pdf Exclusion reasons: Q5 [III.ajk]This paper does not aspire to empiricity.
289. Egon Börger & Robert Stärk (2004): Exploiting Abstraction for Specification Reuse. The Java/C# Case Study. Volume 3188.In de Boer, Frank and Bonsangue, Marcello and Graf, Susanne and de Roever, Willem-Paul (ed.) Formal Methods for Components and Objects.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 42-76. doi:10.1007/978-3-540-30101-1_3 Exclusion reasons: Q5 [III.ajk]This article presents a formal theoretical study of these languages; there is no empiricality involved.
290. Martin Büchi & Wolfgang Weck (2000): Generic Wrappers. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 201-225. doi:10.1007/3-540-45102-1_10 Exclusion reasons: Q5 [II.ajk]Conceptual and theoretical work.
291. Bruno Cabral & Paulo Marques (2007): Exception Handling: A Field Study in Java and .NET. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 151-175. doi:10.1007/978-3-540-73589-2_8 Exclusion reasons: Q1–2 [III.ajk]This article examines empirically the actual usage of exception handling in .NET and Java programs. While this study undoubtedly would be informative to a programming language designer, it does not evaluate design decisions.
292. J. Cai & R. Paige (1987): Binding performance at language design time. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 85-97. doi:10.1145/41625.41633 Exclusion reasons: Q1–2 [III.ajk]This theoretical article does not evaluate language designs.
293. Jiazhen Cai & Robert A. Paige (1991): Look ma, no hashing, and no arrays neither. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 143-154. doi:10.1145/99583.99605 Exclusion reasons: Q1–2 [III.ajk]This article has no relevance to language design.
294. Paul Calder & Mark Linton (1992): The object-oriented implementation of a document editor. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications. Pages 154-165. doi:10.1145/141936.141950 Exclusion reasons: Q1–2 [III.ajk]This article has no PL design relevance.
295. Brad Calder, Dirk Grunwald & Benjamin Zorn (1994): Quantifying behavioral differences between C and C++ programs. Journal of Programming Languages 2 (4). http://cseweb.ucsd.edu/users/calder/abstracts/C++Study.html Exclusion reasons: Q1–2 [II.ajk]This article discusses implementation concerns only.
296. Oscar Callaú, Romain Robbes, Éric Tanter & David Röthlisberger (2011): How developers use the dynamic features of programming languages: the case of smalltalk. In Proceedings of the 8th Working Conference on Mining Software Repositories. New York, NY, USA: ACM. MSR '11. Pages 23-32. doi:10.1145/1985441.1985448 Exclusion reasons: Q1–2 [II.ajk]This article studies actual language usage and does not evaluate any language design decisions.
297. Robert D. Cameron (1989): Efficient high-level iteration with accumulators. ACM Transactions on Programming Languages and Systems 11 (2). Pages 194-211. doi:10.1145/63264.63401 Exclusion reasons: Q5 [III.ajk]This paper does not aspire to empiricity.
298. A. Caracciolo (1966): Some preliminary remarks on theoretical pragmatics. Communications of the ACM 9 (3). Pages 226-227. doi:10.1145/365230.365273 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
299. L. Cardelli (1988): Structural subtyping and the notion of power type. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 70-79. doi:10.1145/73560.73566 Exclusion reasons: Q5 [III.ajk]This analytical paper has no aspiration to empiricity.
300. L. Cardelli, J. Donahue, M. Jordan, B. Kalsow & G. Nelson (1989): The Modula–3 type system. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 202-212. doi:10.1145/75277.75295 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
301. Luca Cardelli (1995): A language with distributed scope. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 286-297. doi:10.1145/199448.199516 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
302. Luca Cardelli (1997): Program fragments, linking, and modularization. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Program-

ming Languages (POPL). Pages 266-277. doi:10.1145/263699.263735 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

303. Luca Cardelli & Andrew D. Gordon (1999): Types for mobile ambients. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 79-92. doi:10.1145/292540.292550 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
304. Jacques Carette (2006): Gaussian Elimination: A case study in efficient genericity with MetaOCaml. Science of Computer Programming 62 (1). Pages 3-24. doi:10.1016/j.scico.2005.10.012 Exclusion reasons: Q1–2 [II.ajk]No comparison.
305. T. T. Carey & M. M. Shepherd (1988): Towards empirical studies of programming in new paradigms. In Proceedings of the 1988 ACM sixteenth annual conference on Computer science. Pages 72-78. doi:10.1145/322609.322618 Exclusion reasons: Q1–2 [III.ajk]This article describes ongoing empirical studies on programmer learning, and presents no PL design issue.
306. T. A. Cargill (1986): Pi: a case study in object-oriented programming. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 350-360. doi:10.1145/28697.28733 Exclusion reasons: Q1–2 [II.ajk]No comparison.
307. Denis Caromel & Julien Vayssière (2001): Reflections on MOP s, Components, and Java Security. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 256-274. doi:10.1007/3-540-45337-7_14 Exclusion reasons: Q5 [III.ajk]This article is analytical, with no empirical aspirations.
308. Denis Caromel, Luis Mateu & Éric Tanter (2004): Sequential Object Monitors. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 317-341. doi:10.1007/978-3-540-24851-4_15 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
309. John W. Carr, III (1959): Recursive subscripting compilers and list-type memories. Communications of the ACM 2 (2). Pages 4-6. doi:10.1145/368280.368281 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
310. Nicholas Carriero, David Gelernter & Jerrold Leichter (1986): Distributed data structures in Linda. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 236-242. doi:10.1145/512644.512666 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
311. Manuel Carro, José F. Morales, Henk L. Muller, G. Puebla & M. Hermenegildo (2006): High-level languages for small devices: a case study. In Proceedings of the 2006 international conference on Compilers, architecture and synthesis for embedded systems. New York, NY, USA: ACM. Pages 271-281. doi:10.1145/1176760.1176794 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
312. Bernard Carré & Jean-Marc Geib (1990): The point of view notion for multiple inheritance. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 312-321. doi:10.1145/97945.97983 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
313. Robert Cartwright, Robert Hood & Philip Matthews (1981): Paths: an abstract alternative to pointers. In Proc. 8th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 14-27. doi:10.1145/567532.567534 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
314. M. Cartwright & M. Shepperd (2000): An empirical investigation of an object-oriented software system . Software Engineering, IEEE Transactions on 26 (8). Pages 786-796. doi:10.1109/32.879814 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which a large existing system was described using various metrics. It does not discuss language design decision efficacy issues.
315. Francisco Heron de Carvalho Junior & Cenez Araújo de Rezende (2012): A case study on expressiveness and performance of component-oriented parallel programming. Journal of Parallel and Distributed Computing. doi:10.1016/j.jpdc.2012.12.007 Exclusion reasons: Q5 [III.ajk]It is not clear whether the construct being evaluated is a programming language, but in any case the evaluation is analytical despite containing performance evaluations.
316. Calin Cascaval, Colin Blundell, Maged Michael, Harold W. Cain, Peng Wu, Stefanie Chiras & Siddhartha Chatterjee (2008): Software transactional memory: why is it only a research toy?. Communications of the ACM 51 (11). Pages 40-46. doi:10.1145/1400214.1400228 Exclusion reasons: Q1–2 [III.ajk]This article discusses implementation efficiency issues.
317. P. Caspi, D. Pilaud, N. Halbwachs & J. A. Plaice (1987): LUSTRE: a declarative language for real-time programming. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 178-188. doi:10.1145/41625.41641 Exclusion reasons: Q1–2 [II.ajk]Language exposition with theoretical emphasis.
318. Paul Caspi, Grégoire Hamon & Marc Pouzet (2007): Synchronous Functional Programming with Lucid Synchrone. In Nicolas Navet and Stephan Mertz (ed.) Real-Time Systems: Models and verification: Theory and tools.ISTE. http://www.di.ens.fr/~pouzet/bib/chap_lucid_synchrone_english_iste08.pdf Exclusion reasons: Q5 [III.ajk]This tutorial does not report a study.
319. Giuseppe Castagna (1995): Covariance and contravariance: conflict without a cause. ACM Transactions on Programming Languages and Systems 17 (3). Pages 431-447. doi:10.1145/203095.203096 Exclusion reasons: Q5 [II.ajk]This theoretical article does not aspire to empiricity.
320. Giuseppe Castagna, Nils Gesbert & Luca Padovani (2009): A theory of contracts for Web services. ACM Transactions on Programming Languages and Systems 31 (5). doi:10.1145/1538917.1538920 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
321. Vincent Cavé, Zoran Budimlić & Vivek Sarkar (2010): Comparing the usability of library vs. language approaches to task parallelism. In Evaluation and Usability of Programming Languages and Tools. New York, NY, USA: ACM. PLATEAU '10. Pages 9:1–9:6. doi:10.1145/1937117.1937126 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
322. José Juan Cañas, Maria Teresa Bajo & Pilar Gonzalvo (1994): Mental models and computer programming. International Journal of Human-Computer Studies 40 (5). Pages 795-811. doi:10.1006/ijhc.1994.1038 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
323. Mariano Ceccato, Paolo Tonella & Filippo Ricca (2005): Is AOP code easier or harder to test than OOP code. In On-line Proceedings of the First Workshop on Testing Aspect-Oriented Programs (WTAOP 2005). http://selab.fbk.eu/tonella/papers/wtaop2005.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
324. Henry Cejtin, Suresh Jagannathan & Richard Kelsey (1995): Higher-order distributed objects. ACM Transactions on Programming Languages and Systems 17 (5). Pages 704-739. doi:10.1145/213978.213986 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate the efficacy of its design decisions.
325. S. Ceri & G. Gottlob (1986): Normalization of relations and PROLOG. Communications of the ACM 29 (6). Pages 524-544. doi:10.1145/5948.5952 Exclusion reasons: Q1–2 [II.ajk]Program exposition.
326. Manuel M. T. Chakravarty, Gabriele Keller, Simon Peyton Jones & Simon Marlow (2005): Associated types with class. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 1-13. doi:10.1145/1040305.1040306 Exclusion reasons: Q5 [II.ajk]Formal type-theoretic work.
327. Bradford L. Chamberlain, Steven J. Deitz & Lawrence Snyder (2000): A comparative study of the NAS MG benchmark across parallel languages and architectures. In Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM). Washington, DC, USA: IEEE Computer Society. Supercomputing '00. Article 46. http://dl.acm.org/citation.cfm?id=370049.370452 Exclusion reasons: Q1–2 [III.ajk]This article reports a study that compares multiple languages. There are no identifiable design decisions at play.
328. Craig Chambers (1992): Object-oriented multi-methods in Cecil. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 33-56. doi:10.1007/BFb0053029 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
329. Craig Chambers (1993): Predicate Classes. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 268-296. doi:10.1007/3-540-47910-4_15 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
330. Craig Chambers & Gary T. Leavens (1995): Typechecking and modules for multimethods. ACM Transactions on Programming Languages and Systems 17 (6). Pages 805-843. doi:10.1145/218570.218571 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
331. Craig Chambers, Bill Harrison & John Vlissides (2000): A debate on language and tool support for design patterns. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 277-289. doi:10.1145/325694.325731 Exclusion reasons: Q1–2Q5 [III.ajk]This article contains position statements and responses; no research is reported.
332. A. T. Chamillard, Lori A. Clarke & George S. Avrunin (1996): An Empirical Comparison of Static Concurrency Analysis Techniques. Technical Report. http://laser.cs.umass.edu/techreports/96-84.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
333. Ashok K. Chandra (1981): Programming primitives for database languages. In Proc. 8th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 50-62. doi:10.1145/567532.567537 Exclusion reasons: Q1–2 [II.ajk]Theoretical study.
334. S. K. Chang, G. Polese, S. Orefice & M. Tucci (1994): A methodology and interactive environment for iconic language design. International Journal of Human-Computer Studies 41 (5). Pages 683-716. doi:10.1006/ijhc.1994.1078 Exclusion reasons: Q1–2 [III.ajk]Visual languages are specifically excluded.
335. Arthur Charlesworth (1987): The multiway rendezvous. ACM Transactions on Programming Languages and Systems 9 (3). Pages 350-366. doi:10.1145/24039.24050 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
336. Baudouin Le Charlier & Pascal Van Hentenryck (1994): Experimental evaluation of a generic abstract interpretation algorithm for PROLOG. ACM

Transactions on Programming Languages and Systems 16 (1). Pages 35-101. doi:10.1145/174625.174627 Exclusion reasons: Q1–2 [II.ajk]Program analysis technique.

337. Sarah Chasins (2011): Efficient implementation of the plaid language. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 209-210. doi:10.1145/2048147.2048211 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision for efficacy.
338. S. Chatel & F. Détienne (1993): Transfer Among Programming Languages: An Assessment of Various Indicators. In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article reports a study of transfer effect from a familiar programming language to an unfamiliar one. Although the experimental setup could be used to answer PL design issue questions, the reported results are not transferable to that domain.
339. Sophie Chatel & Françoise Détienne (1996): Strategies in object-oriented design. Acta Psychologica 91 (3). Pages 245-269. doi:10.1016/0001-6918(95) 00058-5 http://www.sciencedirect.com/science/article/pii/0001691895000585 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
340. Jarnie Chattratichart & Jasna Kuljis (2000): An assessment of visual representations for the 'flow of control'. In PPIG 2000. (Found in http://ppig.org/workshops/12th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article deals with nontextual programming, which is excluded under our protocol.
341. Jarinee Chattratichart & Jasna Kuljis (2000): A Comprehensibility Comparison of Three Visual Representations and a Textual Program in Two Paradigms. Paper in Citeseer. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.87.434 Exclusion reasons: Q1–2 [III.ajk]This article compares a textual program to visual programs. Since visual languages are excluded, there is no admissible language design decision at play.
342. B. D. Chaudhary & H. V. Sahasrabuddhe (1985): A study in dimensions of psychological complexity of programs. International Journal of Man-Machine Studies 23 (2). Pages 113-133. doi:10.1016/S0020-7373(85)80028-6 Exclusion reasons: Q1–2 [II.ajk]Based on the abstract, this article does not evaluate language design decisions.
343. T. E. Cheatham, Jr. (1963): Programming languages. Communications of the ACM 6 (7). Pages 391-395. doi:10.1145/366663.383390 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
344. Marina C. Chen (1986): A parallel language and its compilation to multiprocessor machines or VLSI. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 131-139. doi:10.1145/512644.512656 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
345. H. Chen (1995): Application of Object-oriented Programming in Simulation: A Simulation of Case Study Using Microsoft Visual C++. Technical Report at Miami University Computer Science and Systems Analysis. http://hdl.handle.net/2374.MIA/258 Exclusion reasons: Q1–2 [II.ajk]No language or construct comparison; no PL relevance.
346. Yaofei Chen (2003): Programming Language Trends: An Empirical Study. . PhD at New Jersey Institute of Technology College of Computer Science. http://library1.njit.edu/etd/2000s/2003/njit-etd2003-106/njit-etd2003-106.html Exclusion reasons: Q1–2 [II.ajk]Stdies language evolution, does not evaluate efficacy.
347. Yaofei Chen, R. Dios, A. Mili, Lan Wu & Kefei Wang (2005): An empirical study of programming language trends. Software, IEEE 22 (3). Pages 72-79. doi:10.1109/MS.2005.55 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
348. James Cheney & Christian Urban (2008): Nominal logic programming. ACM Transactions on Programming Languages and Systems 30 (4). doi: 10.1145/1387673.1387675 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical study.
349. Wan-Hong S. Cheng & Virgil E. Wallentine (1989): DEBL: a knowledge-based language for specifying and debugging distributed programs. Communications of the ACM 32 (9). Pages 1079-1084. doi:10.1145/66451.66455 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
350. Yoonsik Cheon & Gary T. Leavens (2002): A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 231-255. doi:10.1007/3-540-47993-7_10 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision, nor does it aspire to empiricity.
351. D. R. Cheriton & M. E. Wolf (1987): Extensions for multi-module records in conventional programming languages. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 296-306. doi:10.1145/41625.41652 Exclusion reasons: Q5 [III.ajk]This paper has no aspiration to empiricity.
352. Shigeru Chiba (2000): Load-Time Structural Reflection in Java. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 313-336. doi:10.1007/3-540-45102-1_16 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
353. Shigeru Chiba & Rei Ishikawa (2005): Aspect-Oriented Programming Beyond Dependency Injection. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 121-143. doi:10.1007/11531142_6 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.
354. SC Chiemeke, KC Ukaoha & SOP Oliomogbe (2006): An empirical comparison of Qbasic, FORTRAN, C, Pascal, C++, Visual Basic and Visual C++. The Information Technologist 3 (1). Pages 63-75. http://www.ajol.info/index.php/ict/article/view/31964 Exclusion reasons: Q5 [III.ajk]This article styles itself as reporting an "empirical" comparison of several programming languages. In the study, a single well-defined algorithm has been implemented (presumably once, presumably by one of the researchers) in each of the languages, and the resulting programs are compared. As the algorithm is well specified, there isn't much freedom left to the implementor, and thus this study is more about the implications of the languages in question (and thus analytical) than about the contingent aspects of the world given the existing designs of the languages. The article is, thus, not empirical as that word is used in this mapping study.
355. Brian Chin & Todd Millstein (2006): Responders: Language Support for Interactive Applications. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 255-278. doi:10.1007/11785477_17 Exclusion reasons: Q5 [III.ajk]This article includes two analytic-constructive "case studies" that are not empirical in nature.
356. Ruzanna Chitchyan, Phil Greenwood, Americo Sampaio, Awais Rashid, Alessandro Garcia & Lyrene Fernandes da Silva (2009): Semantic vs. syntactic compositions in aspect-oriented requirements engineering: an empirical study. In Proceedings of the 8th ACM international conference on Aspect-oriented software development. New York, NY, USA: ACM. Pages 149-160. doi:10.1145/1509239.1509260 Exclusion reasons: Q1–2 [III.ajk]This article deals with requirements, not programming.
357. Woojin Choi & J. Draper (2011): Unified Signatures for Improving Performance in Transactional Memory. In Parallel Distributed Processing Symposium (IPDPS), 2011 IEEE International. Pages 817-827. doi:10.1109/IPDPS.2011.81 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
358. R. Christ, S. L. Halter, K. Lynne, S. Meizer, S. J. Munroe & M. Pasch (2000): San Francisco performance: A case study in performance for large-scale Java applications. IBM Systems Journal 39 (1). Pages 4-20. doi:10.1147/sj.391.0004 Exclusion reasons: Q1–2 [II.ajk]No PL comparison.
359. Aske Simon Christensen, Anders Møller & Michael I. Schwartzbach (2003): Extending Java for high-level Web service construction. ACM Transactions on Programming Languages and Systems 25 (6). Pages 814-875. doi:10.1145/945885.945890 Exclusion reasons: Q5 [III.ajk]This is a largely analytic-constructive paper. The only part that is arguably empirical evaluates the implementation with no comparison. With respect to language design, the paper has no aspiration to empiricity.
360. Yaohan Chu (1965): An ALGOL-like computer design language. Communications of the ACM 8 (10). Pages 607-615. doi:10.1145/365628.365650 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
361. Luke Church, Chris Nash & Alan Blackwell (2010): Liveness in Notation Use: From Music to Programming Spreadsheet Users?. In PPIG 2010. (Found in http://ppig2010.org/index.php?title=Program.) http://ppig.org/papers/22nd-UX-1.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.
362. Douglas W. Clark & C. Cordell Green (1977): An empirical study of list structure in Lisp. Communications of the ACM 20 (2). Pages 78-87. doi: 10.1145/359423.359427 Exclusion reasons: Q1–2 [II.ajk]Study of data patterns Lisp programs.
363. Lawrence Clark (1984): A linguistic contribution to GOTO-less programming. Communications of the ACM 27 (4). Pages 349-350. doi:10.1145/358027.358043 Exclusion reasons: Q1–2 [III.ajk]This article is a satirical exposition of a purported language feature.
364. Keith Clark & Steve Gregory (1986): PARLOG: parallel programming in logic. ACM Transactions on Programming Languages and Systems 8 (1). Pages 1-49. doi:10.1145/5001.5390 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
365. Tony Clark & Laurence Tratt (2009): Language factories. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 949-955. doi:10.1145/1639950.1640062 Exclusion reasons: Q1–2 [II.ajk]Abstract indicates this is a position paper about how to develop DSLs more systematically than currently.
366. David G. Clarke, James Noble & John M. Potter (2001): Simple Ownership Types for Object Containment. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 53-76. doi:10.1007/3-540-45337-7_4 Exclusion reasons: Q5 [II.ajk]Formal discussion.

367. Steven Clarke (2001): Evaluating a new programming language. In PPIG 2001. (Found in http://ppig.org/workshops/13th-programme.html.) Exclusion reasons: Q3 [III.ajk]This article reports a study evaluating a language evaluation method. Evaluating language design decisions is a part of that but not the focus, and viewed as a language design decision evaluation, it is insufficiently reported here.
368. Dave Clarke & Tobias Wrigstad (2003): External Uniqueness Is Unique Enough. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 176-201. doi:10.1007/978-3-540-45070-2_9 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.
369. Jens Claßen, Viktor Engelmann, Gerhard Lakemeyer & Gabriele Röger (2008): Integrating Golog and Planning: An Empirical Evaluation. In Maurice Pagnucco and Michael Thielscher (ed.) Proceedings of the Twelfth International Workshop on Non-Monotonic Reasoning. Pages 10-18. http://www.cse.unsw.edu.au/~kr2008/NMR2008/ Exclusion reasons: Q1–2 [III.ajk]If I understand this article correctly, it is comparing implementation techniques for a single language, not language design alternatives.
370. Romain E. Cledat, Tushar Kumar & Santosh Pande (2011): Efficiently speeding up sequential computation through the n-way programming model. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 537-554. doi:10.1145/2048066.2048109 Exclusion reasons: Q1–2 [III.ajk]This article only evaluates performance with no relevance to efficacy.
371. Geoffrey Clemm & Leon Osterweil (1990): A mechanism for environment integration. ACM Transactions on Programming Languages and Systems 12 (1). Pages 1-25. doi:10.1145/77606.77607 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
372. Curtis Clifton, Todd Millstein, Gary T. Leavens & Craig Chambers (2006): MultiJava: Design rationale, compiler implementation, and applications. ACM Transactions on Programming Languages and Systems 28 (3). Pages 517-575. doi:10.1145/1133651.1133655 Exclusion reasons: Q3 [III.ajk]This article does include in Section 5 a report of usage in the wild and received user feedback, which could be generously interpreted as an empirical study on efficacy; however, the methodology of that study is not described at all.
373. Curtis Clifton, Gary T. Leavens & James Noble (2007): MAO: Ownership and Effects for More Effective Reasoning About Aspects. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609. Pages 451-475. doi:10.1007/978-3-540-73589-2_22 Exclusion reasons: Q3 [III.ajk]This article includes a self-styled "case study" that claims to provide "preliminary indications" in favour of the efficacy of the proposed construct. What was done, exactly, remains unclear to me, especially considering that the data files linked to in the article have disappeared from the web.
374. W. D. Climenson (1963): RECOL - a retrieval command language. Communications of the ACM 6 (3). Pages 117-122. doi:10.1145/366274.366342 Exclusion reasons: Q1–2 [II.ajk]Language exposition only.
375. William Clinger & Jonathan Rees (1991): Macros that work. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 155-162. doi:10.1145/99583.99607 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
376. Cristian Coarfa, Yuri Dotsenko, Jason Eckhardt & John Mellor-Crummey (2004): Co-array Fortran Performance and Potential: An NPB Experimental Study. Volume 2958.In Rauchwerger, Lawrence (ed.) Languages and Compilers for Parallel Computing.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 177-193. doi:10.1007/978-3-540-24644-2_12 Exclusion reasons: Q1–2 [II.ajk]Implementation issue.
377. Richard Cobbe & Matthias Felleisen (2005): Environmental acquisition revisited. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 14-25. doi:10.1145/1040305.1040307 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
378. E. F. Codd, E. S. Lowry, E. McDonough & C. A. Scalzi (1959): Multiprogramming STRETCH: feasibility considerations. Communications of the ACM 2 (11). Pages 13-17. doi:10.1145/368481.368502 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
379. Roberta Coelho, O Lemos, F Ferrari, A Staa & P Masiero (2009): On the robustness assessment of aspect oriented programs. In Proceedings of Third Workshop on Assessment of Contemporary Modularization Techniques, Co-located with the 24th ACM Conference on Object-Oriented Programming Systems and Applications, Orlando, Florida, USA. http://www.comp.lancs.ac.uk/~greenwop/ACoM.09/coelho.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
380. Kenneth Cohen & J. H. Wegstein (1965): AXLE2: an axiomatic language for string transformations. Communications of the ACM 8 (11). Pages 657-661. doi:10.1145/365660.365669 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
381. Jacques Cohen & Carl Zuckerman (1974): Two languages for estimating program efficiency. Communications of the ACM 17 (6). Pages 301-308. doi:10.1145/355616.361015 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
382. Jacques Cohen & Timothy J. Hickey (1987): Parsing and compiling using Prolog. ACM Transactions on Programming Languages and Systems 9 (2). Pages 125-163. doi:10.1145/22719.22946 Exclusion reasons: Q1–2 [II.ajk]Language usage tutorial, no PL design relevance.
383. Jacques Cohen (1988): A view of the origins and development of Prolog. Communications of the ACM 31 (1). Pages 26-36. doi:10.1145/35043.35045 Exclusion reasons: Q1–2 [III.ajk]This article does not report a design-evaluative study.
384. Jacques Cohen (1990): Constraint logic programming languages. Communications of the ACM 33 (7). Pages 52-68. doi:10.1145/79204.79209 Exclusion reasons: Q1–2 [III.ajk]This tutorial article does not report an evaluative study.
385. Tal Cohen & Joseph (Yossi) Gil (2004): AspectJ2EE = AOP + J2EE: Towards an Aspect Based, Programmable, and Extensible Middleware Framework. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 221-245. doi:10.1007/978-3-540-24851-4_10 Exclusion reasons: Q5 [III.ajk]This article introduces and discusses a new construction; it has no aspiration for empiricity.
386. Jerald D. Cole (1991): WHILE loops and the analogy of the single stroke engine. SIGCSE Bulletin 23 (3). Pages 20-22. doi:10.1145/126459.126466 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
387. Christian Collberg, Clark Thomborson & Douglas Low (1998): Manufacturing cheap, resilient, and stealthy opaque constructs. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 184-196. doi:10.1145/268946.268962 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
388. Trevor Collins & Pat Fung (1999): Evaluating Hank. In PPIG 1999. (Found in http://ppig.org/workshops/11th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]The language under evaluation is a visual language, and is excluded under our definition of PLs.
389. Melvin E. Conway (1958): Proposal for an UNCOL. Communications of the ACM 1 (10). Pages 5-8. doi:10.1145/368924.368928 Exclusion reasons: Q5 [III.ajk]This article is a language proposal with no empirical evaluation.
390. Melvin E. Conway (1961): Letters to the editor: ALGOL 60 comment. Communications of the ACM 4 (10). Pages 465. doi:10.1145/366786.366810 Exclusion reasons: Q5 [III.ajk]This letter to the editor does not report an empirical study.
391. R. W. Conway & W. L. Maxwell (1963): CORC - the Cornell computing language. Communications of the ACM 6 (6). Pages 317-321. doi:10.1145/366604.366651 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition together with a report of experience. It does not evaluate design decisions.
392. R. W. Conway, J. J. Delfausse, W. L. Maxwell & W. E. Walker (1965): CLP-the Cornell list processor. Communications of the ACM 8 (4). Pages 215-216. doi:10.1145/363831.363840 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
393. Robert P. Cook & Nitin Donde (1982): An experiment to improve operand addressing. In Proceedings of the first international symposium on Architectural support for programming languages and operating systems. Pages 87-91. doi:10.1145/800050.801830 Exclusion reasons: Q1–2 [III.ajk]This article discusses a language meant for automatic generation and thus is excluded under our protocol.
394. Curtis Cook, William Bregar & David Foote (1984): A preliminary investigation of the use of the cloze procedure as a measure of program understanding. Information Processing & Management 20 (1–2). Pages 199-208. doi:10.1016/0306-4573(84)90050-5 Exclusion reasons: Q1–2 [II.ajk]This article investigates comprehension measurement and does not evaluate any language design decisions.
395. R. P. Cook (1989): An empirical analysis of the Lilith instruction set. Computers, IEEE Transactions on 38 (1). Pages 156-158. doi:10.1109/12.8740 Exclusion reasons: Q1–2 [III.ajk]The language studied is not intended for manual writing, and hence is excluded under our definition of programming languages.
396. Daniel E. Cooke, Brad Nemanich & J. Nelson Rushton (2006): The Role of Theory and Experiment in Language Design–A 15 Year Perspective. In Tools with Artificial Intelligence, 2006. ICTAI '06. 18th IEEE International Conference on. Pages 163-168. doi:10.1109/ICTAI.2006.112 Exclusion reasons: Q6 Q7 [III.ajk]This retrospective article does not deal with empirical evidence. Although it does throw the word "experiment" around, those "experiments" appear to be analytical-constructive explorations of the implications of an existing construct and thus not empirical in our sense.
397. Daniel E. Cooke, J. Nelson Rushton, Brad Nemanich, Robert G. Watson & Per Andersen (2008): Normalize, transpose, and distribute: An automatic approach for handling nonscalars. ACM Transactions on Programming Languages and Systems 30 (2). doi:10.1145/1330017.1330020 Exclusion reasons: Q1–2 Disagreement resolution result. [sel-2.kaijanaho]No empiricity. [sel-2.tirronen]Presents a language, does not measure other than by telling authors experiences. Experiences only.
398. M. J. Coombs, R. Gibson & J. L. Alty (1982): Learning a first computer language: strategies for making sense. International Journal of Man-Machine Studies 16 (4). Pages 449-486. doi:10.1016/S0020-7373(82)80051-5 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate language design decisions.
399. J. Cordsen, J. Nolte & W. Schröder-Preikschat (1998): Experiences developing a virtual shared memory system using high-level object paradigms.

In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 285-306. doi:10.1007/BFb0054096 Exclusion reasons: Q1–2 [II.ajk]No PL relevance.

400. Antonio Corradi, Letizia Leonardi & Franco Zambonelli (2001): Parallel object allocation via user-specified directives: A case study in traffic simulation. Parallel Computing 27 (3). Pages 223-241. doi:10.1016/S0167-8191(00)00105-8 http://www.sciencedirect.com/science/article/pii/S0167819100001058 Exclusion reasons: Q1–2 [III.ajk]In this study, there is no comparison design.
401. C. L. Corritore & S. Wiedenbeck (2000): Direction and scope of comprehension-related activities by procedural and object-oriented programmers: an empirical study. In Program Comprehension, 2000. Proceedings. IWPC 2000. 8th International Workshop on. Pages 139-148. doi:10.1109/WPC.2000.852488 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
402. CYNTHIA L. CORRITORE & SUSAN WIEDENBECK (2001): An exploratory study of program comprehension strategies of procedural and object-oriented programmers. International Journal of Human-Computer Studies 54 (1). Pages 1-23. doi:10.1006/ijhc.2000.0423 Exclusion reasons: Q1–2 [II.ajk]This article seems not to evaluate language design decisions.
403. Corinna Cortes, Kathleen Fisher, Daryl Pregibon, Anne Rogers & Frederick Smith (2004): Hancock: A language for analyzing transactional data streams. ACM Transactions on Programming Languages and Systems 26 (2). Pages 301-338. doi:10.1145/973097.973100 Exclusion reasons: Q1–2 [III.ajk]This language exposition paper does not evaluate its construct.
404. John Corwin, David F. Bacon, David Grove & Chet Murthy (2003): MJ: a rational module system for Java and its applications. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. Pages 241-254. doi:10.1145/949305.949326 Exclusion reasons: Q5 [III.ajk]This article contains an "experiment" (as the article calls it) which consists of the authors applying their new system to an existing program. That is exploration of the implications of the construct and therefore not empirical.
405. Thomas Cottenier, Aswin van den Berg & Tzilla Elrad (2007): Joinpoint Inference from Behavioral Specification to Implementation. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609. Pages 476-500. doi:10.1007/978-3-540-73589-2_23 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
406. Neal S. Coulter & Norman H. Kelly (1986): Computer instruction set usage by programmers: an empirical investigation. Communications of the ACM 29 (7). Pages 643-647. doi:10.1145/6138.6148 Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study about usage patterns of assembly language opcodes in actual human-written code. It does not evaluate a design decision.
407. S. Counsell & P. Newson (2000): Use of friends in C++ software: an empirical investigation. Journal of Systems and Software 53 (1). Pages 15-21. doi:10.1016/S0164-1212(00)00004-2 Exclusion reasons: Q1–2 [II.ajk]Feature usage study, no language or feature comparison.
408. Michael A. Covington (1984): A pedagogical disadvantage of repeat and while. SIGPLAN Notices 19 (8). Pages 85-86. doi:10.1145/988241.988247 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
409. Anthony Cox, Maryanne Fisher, Diana Smith & Josipa Granic (2004): Learning and using formal language. In PPIG 2004. (Found in http://ppig.org/workshops/16th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.
410. S. Crespi-Reghizzi & R. Morpurgo (1970): A language for treating graphs. Communications of the ACM 13 (5). Pages 319-323. doi:10.1145/362349.362366 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
411. S. Crespi-Reghizzi, M. A. Melkanoff & L. Lichten (1973): The use of grammatical inference for designing programming languages. Communications of the ACM 16 (2). Pages 83-90. doi:10.1145/361952.361958 Exclusion reasons: Q1–2 [II.ajk]Grammar discovery; no PL design issue.
412. Cyrus .J. Creveling (1968): Experimental Use of A Programming Language (APL) at the Goddard Space Flight Center.. at National Aeronautics and Space Administration, Goddard Space Flight Center. http://eric.ed.gov/ERICWebPortal/detail?accno=ED067251 Exclusion reasons: Q1–2 [II.ajk]No comparison.
413. Adrienne Critcher (1979): The functional power of parameter passage mechanism. In Proc. 6th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 158-168. doi:10.1145/567752.567767 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
414. Adrienne Critcher (1982): On the ability of structures to store and access information. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 366-378. doi:10.1145/582153.582191 Exclusion reasons: Q5 Q7 [III.ajk]This is an analytic paper with no aspiration to empiricity.
415. Lobel Crnogorac, Anand S. Rao & Kotagiri Ramamohanarao (1998): Classifying inheritance mechanisms in concurrent object-oriented programming. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 571-600. doi:10.1007/BFb0054108 Exclusion reasons: Q5 [II.ajk]Formal theoretical study.
416. Joseph W. Croghan, Myron L. Cramer & Joan Hardy (1990): Implementing advanced artificial intelligence concepts in Ada: a case study of a prototype expert system for a real-time electronic warfare application. In Proceedings of the seventh Washington Ada symposium on Ada. New York, NY, USA: ACM. Pages 255-259. doi:10.1145/327011.327115 Exclusion reasons: Q1–2 [III.ajk]This article reports on a software project; there was no PL design issue involved.
417. D. Crookes & J. W. G. Elder (1984): An experiment in language design for distributed systems. Software: Practice and Experience 14 (10). Pages 957-971. doi:10.1002/spe.4380141006 Exclusion reasons: Q3 [III.ajk]This article is a language exposition with an analytical approach. There is a brief section about evaluation but it contains few details as to its actual methodology. No fuller report has been identified.
418. M. E. Crosby & J. Stelovsky (1990): How do we read algorithms? A case study. Computer 23 (1). Pages 25-35. doi:10.1109/2.48797 Exclusion reasons: Q1–2 [II.ajk]Study of programmer behaviour. No PL design issue.
419. Martha E. Crosby, Jean Scholtz & Susan Wiedenbeck (2002): The Roles Beacons Play in Comprehension for Novice and Expert Programmers. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
420. L. A. Crowl (1988): Shared memory multiprocessors and sequential programming languages: a case study. Volume 2.In System Sciences, 1988. Vol.II. Software Track, Proceedings of the Twenty-First Annual Hawaii International Conference on. Pages 103-108. doi:10.1109/HICSS.1988.11795 Exclusion reasons: Q1–2 [II.ajk]No comparison.
421. Lawrence A. Crowl & Thomas J. LeBlanc (1994): Parallel programming with control abstraction. ACM Transactions on Programming Languages and Systems 16 (3). Pages 524-576. doi:10.1145/177492.177584 Exclusion reasons: Q5 [III.ajk]This article contains two empirical sections. The first is Section 4.2, in which performance measurements are made regarding an example program. Given its nature as an example, it does not bear on the efficacy question. The second is Section 5.3 evaluating the implementation discussed in Section 5; its stated goal is to evaluate the implementation, and thus also does not bear on the efficacy question.
422. José A. Cruz-Lemus, Marcela Genero, M. Esperanza Manso, Sandro Morasca & Mario Piattini (2009): Assessing the understandability of UML statechart diagrams with composite states—A family of empirical studies. Empirical Software Engineering 14 (6). Pages 685-719. doi:10.1007/s10664-009-9106-z Exclusion reasons: Q1–2 [II.ajk]A graphical language is not within our PL definition.
423. David E. Culler, Seth Copen Goldstein, Klaus Erik Schauser & Thorsten von~Eicken (1992): Empirical Study of a Dataflow Language on the CM-5. In Proc. of the Dataflow Workshop, 19th Int'l Symposium on Computer Architecture. Pages 187-210. http://www.cs.cmu.edu/~seth/papers/culler-wdc92.pdf Exclusion reasons: Q1–2 [III.ajk]Evaluated based on http://reference.kfupm.edu.sa/content/e/i/eicken__empirical_study_of_a_-dataflow_la_97680.pdf – this article is concerned with implementation efficiency on a particular machine type, only.
424. Joseph F. Cunningham (1963): COBOL. Communications of the ACM 6 (3). Pages 79-82. doi:10.1145/366274.366290 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
425. Tom Van Cutsem, Alexandre Bergel, Stéphane Ducasse & Wolfgang De Meuter (2009): Adding State and Visibility Control to Traits Using Lexical Nesting. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 220-243. doi:10.1007/978-3-642-03013-0_11 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
426. Krzysztof Czarnecki & Ulrich W. Eisenecker (1999): Synthesizing Objects. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 18-42. doi:10.1007/3-540-48743-3_2 Exclusion reasons: Q1–2 [II.ajk]Uses the language as a tool, not as an evaluee.
427. Ole-Johan Dahl & Kristen Nygaard (1966): SIMULA: an ALGOL-based simulation language. Communications of the ACM 9 (9). Pages 671-678. doi:10.1145/365813.365819 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
428. J. Dalbey & M. C. Linn (1985): The demands and requirements of computer programming: A literature review. Journal of Educational Computing Research 1 (3). Pages 253–274. doi:10.2190/BC76-8479-YM0X-7FUA Exclusion reasons: Q1–2 [III.ajk]This article summarizes research on teaching programming.
429. J. Daly, J. Miller, A. Brooks, M. Roper & M. Wood (1996): An empirical evaluation of object-oriented practioners' experiences. In Empirical studies of programmers: sixth workshop.Ablex. Pages 267. Exclusion reasons: Q1–2 [III.ajk]More detail about the study is available in refs Daly et al 1995a and Daly 1995b (both available via http://personal.cis.strath.ac.uk/~murray/efocswww/reports.html). The study focuses on object-oriented programming

at a fairly abstract level. Although specific languages are mentioned, the study does not present a PL design issue.

430. Scott Danforth & Ira R. Forman (1994): Reflections on metaclass programming in SOM. In OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications. Pages 440-452. doi:10.1145/191080.191149 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
431. Daniel S. Dantas & David Walker (2006): Harmless advice. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 383-396. doi:10.1145/1111037.1111071 Exclusion reasons: Q3 Q5 [III.ajk]This article mostly does not aspire to empiricity, and the "case studies" are reported very vaguely.
432. Daniel S. Dantas, David Walker, Geoffrey Washburn & Stephanie Weirich (2008): AspectML: A polymorphic aspect-oriented functional programming language. ACM Transactions on Programming Languages and Systems 30 (3). doi:10.1145/1353445.1353448 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
433. F. Dantas (2011): Reuse vs. maintainability: revealing the impact of composition code properties. In Software Engineering (ICSE), 2011 33rd International Conference on. Pages 1082 -1085. doi:10.1145/1985793.1986001 Exclusion reasons: Q1–2 [III.ajk]This article presents a research proposal and does not report a (completed) study.
434. Jared L. Darlington (1990): Search direction by goal failure in goal-oriented programming. ACM Transactions on Programming Languages and Systems 12 (2). Pages 224-252. doi:10.1145/78942.78946 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
435. Eva Darulova & Viktor Kuncak (2011): Trustworthy numerical computation in Scala. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 325-344. doi:10.1145/2048066.2048094 Exclusion reasons: Q1–2 [III.ajk]A library can sometimes seen as a language extension; it is quite true of numerical types that could easily be built in a language and are only left out for prudential reasons or because the types in question are newer than the language. For that reason, this article concerns itself with a language design decision, but it does not evaluate its efficacy in any meaningful way; the only evaluations are about technical performance, with no control comparison provided.
436. John M. Daughtry & John M. Carroll (2010): Perceived Self-Efficacy and APIs. In PPIG 2010. (Found in http://ppig2010.org/index.php?title=Program.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decision efficacy.
437. Cristina David & Wei-Ngan Chin (2011): Immutable specifications for more concise and precise verification. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 359-374. doi:10.1145/2048066.2048096 Exclusion reasons: Q1–2 [III.ajk]This article evaluates its construct only by verification overhead measurements; there is no efficacy question presented.
438. Simon P. Davies (1993): Models and theories of programming strategy. International Journal of Man-Machine Studies 39 (2). Pages 237-267. doi:10.1006/imms.1993.1061 Exclusion reasons: Q1–2 [III.ajk]This article reviews empirical literature on program comprehension and generation strategy but has no direct relevance on language design evaluation.
439. J. Steve Davis (1989): Usability of SQL and menus for database query. International Journal of Man-Machine Studies 30 (4). Pages 447-455. doi:10.1016/S0020-7373(89)80027-6 Exclusion reasons: Q1–2 [II.ajk]Both SQL and menus are excluded from being considered programming languages here.
440. Kei Davis, Yannis Smaragdakis & Jörg Striegnitz (2002): Multiparadigm Programming with Object-Oriented Languages. In ECOOP 2002 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 2548. Pages 154-159. doi:10.1007/3-540-36208-8_13 Exclusion reasons: Q1–2 [III.ajk]This workshop summary does not report a study.
441. Arnab De & Deepak D'Souza (2012): Scalable Flow-Sensitive Pointer Analysis for Java with Strong Updates. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 665-687. doi:10.1007/978-3-642-31057-7_29 Exclusion reasons: Q1–2 [III.ajk]While sensitivity improvements may have efficacy relevance, the connection is too weak to serve as a basis for inclusion.
442. Antonio Wendell De Oliveira Rodrigues, Frédéric Guyomarc'h & Jean-Luc Dekeyser (2011): Programming Massively Parallel Architectures using MARTE: a Case Study. In 2nd Workshop on Model Based Engineering for Embedded Systems Design (M-BED 2011) on Date Conference 2011. http://hal.inria.fr/inria-00578646/en/ Exclusion reasons: Q1–2 [III.ajk]The language being (perhaps) evaluated is a variant of UML and thus not textual.
443. Adrienne Decker (2003): A tale of two paradigms. Journal of Computing Sciences in Colleges 19 (2). Pages 238-246. http://dl.acm.org/citation.cfm?id=948785.948820 Exclusion reasons: Q1–2 [III.ajk]This article studies teaching strategy, not language design decisions.
444. Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx, Theo D'Hondt & Wolfgang De Meuter (2005): Ambient-oriented programming. In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 31-40. doi:10.1145/1094855.1094867 Exclusion reasons: Q1–2 [II.ajk]Exploration of the design space; no evaluation.
445. Jessie Dedecker, Tom Van Cutsem, Stijn Mostinckx, Theo D'Hondt & Wolfgang De Meuter (2006): Ambient-Oriented Programming in AmbientTalk. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 230-254. doi:10.1007/11785477_16 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
446. Pierpaolo Degano, Corrado Priami, Lone Leth & Bent Thomsen (1997): Analysis of Facile programs: A case study. Volume 1192.In Dam, Mads (ed.) Analysis and Verification of Multiple-Agent Languages.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 345-369. doi:10.1007/3-540-62503-8_16 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
447. Markus Degen, Peter Thiemann & Stefan Wehr (2007): Tracking Linear and Affine Resources with Java(X). In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609. Pages 550-574. doi:10.1007/978-3-540-73589-2_26 Exclusion reasons: Q5 [II.ajk]Formal development of a feature.
448. Anthony H. Dekker (1994): The game of life: a CLEAN programming tutorial and case study. SIGPLAN Notices 29 (9). Pages 91-114. doi:10.1145/185009.185032 Exclusion reasons: Q5 [III.ajk]This tutorial article has no empirical content.
449. D. P. Delorey, C. D. Knutson & S. Chun (2007): Do Programming Languages Affect Productivity? A Case Study Using Data from Open Source Projects. In Emerging Trends in FLOSS Research and Development, 2007. FLOSS '07. First International Workshop on. Pages 8. doi:10.1109/FLOSS.2007.5 Exclusion reasons: Q1–2 [III.ajk]This article reports a study that aims to determine whether the choice of a programming language affects programmer productivity. It has no relevance to programming language design, except so far as it motivates it.
450. Daniel P. Delorey, Charles D. Knutson & Mark Davies (2009): Mining Programming Language Vocabularies from Source Code. In PPIG 2009. (Found in http://ppig.org/workshops/21st-programme.html.) Exclusion reasons: Q5 [III.ajk]This empirical study is concerned with language usage patterns, and does not evaluate design decisions.
451. Alan Demers, James Donahue & Glenn Skinner (1978): Data types as values: polymorphism, type-checking, encapsulation. In Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 23-30. doi:10.1145/512760.512764 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
452. Alan Demers & James Donahue (1980): "Type-completeness" as a language principle. In Proc. 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 234-244. doi:10.1145/567446.567469 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
453. Camil Demetrescu, Irene Finocchi & Andrea Ribichini (2011): Reactive imperative programming with dataflow constraints. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 407-426. doi:10.1145/2048066.2048100 Exclusion reasons: Q5 [III.ajk]Empirical evaluation focuses on performance only.
454. Linda G. DeMichiel & Richard P. Gabriel (1987): The Common Lisp Object System: An Overview. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 151-170. doi:10.1007/3-540-47891-4_15 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
455. Rick DeNatale, Charles Irby, John LaLonde, Burton Leathers & Reed Phillips (1990): OOP in the real world. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 299-302. doi:10.1145/97945.97981 Exclusion reasons: Q1–2 [III.ajk]This article does not report a study.
456. S. V. Denisenko (1988): Quantitative evaluation of the efficiency of static semantic program verification. Programming and computer software 14 (3). Pages 143-150. Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
457. Pierre-Malo Deniélou & Nobuko Yoshida (2011): Dynamic multirole session types. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 435-446. doi:10.1145/1926385.1926435 Exclusion reasons: Q5 [II.ajk]Formal type theory development, and some implementation discussion.
458. B. T. Denvir (1979): On orthogonality in programming languages. SIGPLAN Notices 14 (7). Pages 18–30. doi:10.1145/954245.954246 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
459. F DETIENNE (1989): A REVIEW OF PSYCHOLOGICAL STUDIES ON THE COMPREHENSION OF COMPUTER-PROGRAMS. TSI-TECHNIQUE ET

SCIENCE INFORMATIQUES 8 (1). Pages 5-20. Exclusion reasons: Q4 [II.ajk]In French despite the English title returned by Web of Science.

460. A. van Deursen (1997): Domain-Specific Languages versus Object-Oriented Frameworks: A Financial Engineering Case Study. In Proceedings Smalltalk and Java in Industry and Academia, STJA'97. Pages 35-39. http://homepages.cwi.nl/~arie/papers/stja97.pdf Exclusion reasons: Q5 [III.ajk]This article is analytical and has no empirical aspirations.
461. Robert B. K. Dewar, Ronald R. Hochsprung & William S. Worley (1969): The IITRAN programming language. Communications of the ACM 12 (10). Pages 569-575. doi:10.1145/363235.363257 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.
462. Robert B. K. Dewar, Arthur Grand, Ssu-Cheng Liu, Jacob T. Schwartz & Edmond Schonberg (1979): Programming by Refinement, as Exemplified by the SETL Representation Sublanguage. ACM Transactions on Programming Languages and Systems 1 (1). Pages 27-49. doi:10.1145/357062.357064 Exclusion reasons: Q1–2 [II.ajk]Language exposition
463. Mariangiola Dezani-Ciancaglini, Dimitris Mostrous, Nobuko Yoshida & Sophia Drossopoulou (2006): Session Types for Object-Oriented Languages. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 328-352. doi:10.1007/11785477_20 Exclusion reasons: Q5 [II.ajk]Formal theoretical study.
464. Mohan Dhawan, Chung-chieh Shan & Vinod Ganapathy (2012): Enhancing JavaScript with Transactions. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 383-408. doi:10.1007/978-3-642-31057-7_18 Exclusion reasons: Q1–2 [III.ajk]The evaluations in this paper mostly try to show that the technology does what it's supposed to do with modest performance cost. There's no insight into efficacy.
465. Dinakar Dhurjati, Sumant Kowshik & Vikram Adve (2006): SAFECode: enforcing alias analysis for weakly typed languages. In Proceedings of the 2006 ACM SIGPLAN conference on Programming language design and implementation. New York, NY, USA: ACM. PLDI '06. Pages 144-157. doi:10.1145/1133981.1133999 Exclusion reasons: Q1–2 [III.ajk]This article studies something that could be characterised as an implementation technique or a static analysis technique but does not really qualify for a language design decision.
466. Ricardo Dias, Dino Distefano, João Seco & João Lourenço (2012): Verification of Snapshot Isolation in Transactional Memory Java Programs. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 640-664. doi:10.1007/978-3-642-31057-7_28 Exclusion reasons: Q1–2 [III.ajk]The (arguably) empirical evaluation focuses only on verification overhead, with no efficacy implications.
467. Sylvia Dieckmann & Urs Hölzle (1999): A study of the Allocation Behavior of the SPECjvm98 Java Benchmarks. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 92-115. doi:10.1007/3-540-48743-3_5 Exclusion reasons: Q1–2 [II.ajk]Program behaviour study, no PL design issue.
468. Larry Ramon Diesen (1968): Some Applications of an Experimental Language for Doing Symbolic Mathematics. at The American University. Exclusion reasons: Q1–2 [III.ajk]This article does not aspire to empiricity; its evaluation of the language is merely analytic in nature.
469. Werner Dietl, Sophia Drossopoulou & Peter Müller (2007): Generic Universe Types. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 28-53. doi:10.1007/978-3-540-73589-2_3 Exclusion reasons: Q1–2 [II.ajk]Type theory formal development, based on the abstract.
470. Werner Dietl, Michael D. Ernst & Peter Müller (2011): Tunable Static Inference for Generic Universe Types. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 333-357. doi:10.1007/978-3-642-22655-7_16 Exclusion reasons: Q1–2 [II.ajk]Implementation issue.
471. Suzanne W. Dietrich (1992): Shortest path by approximation in logic programs. ACM Transactions on Programming Languages and Systems 1 (2). Pages 119-137. doi:10.1145/151333.151377 Exclusion reasons: Q5 [III.ajk]This paper discusses algorithm issues in light of changes in language semantics. However, the approach is analytical, not empirical.
472. Jens Dietrich, Catherine McCartin, Ewan Tempero & Syed Shah (2010): Barriers to Modularity - An Empirical Study to Assess the Potential for Modularisation of Java Programs. Volume 6093.In Heineman, George and Kofron, Jan and Plasil, Frantisek (ed.) Research into Practice – Reality and Gaps. Lecture Notes in Computer Science. Pages 135-150. doi:10.1007/978-3-642-13821-8_11 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
473. Danny Dig, John Marrero & Michael D. Ernst (2011): How do programs become more concurrent: a story of program transformations. In Proceedings of the 4th International Workshop on Multicore Software Engineering. New York, NY, USA: ACM. IWMSE '11. Pages 43-50. doi:10.1145/1984693.1984700 Exclusion reasons: Q1–2 [II.ajk]This article examines program evolution, not any language design decisions.
474. E. W. Dijkstra (1961): Letter to the editor: defense of ALGOL 60. Communications of the ACM 4 (11). Pages 502-503. doi:10.1145/366813.366844 Exclusion reasons: Q1–2 [III.ajk]This letter to the editor does not report a study.
475. E. W. Dijkstra (1965): Programming Considered as a Human Activity. In Proc. IFIP Congress I. Exclusion reasons: Q5 Q6 [III.ajk]This article does not aspire to empiricity.
476. Edsger W. Dijkstra (1968): Letters to the editor: go to statement considered harmful. Communications of the ACM 11 (3). Pages 147-148. doi:10.1145/362929.362947 Exclusion reasons: Q5 [III.ajk]This famous letter to the editor does not aspire to empiricity.
477. Edsger W. Dijkstra (1975): Guarded commands, nondeterminacy and formal derivation of programs. Communications of the ACM 18 (8). Pages 453-457. doi:10.1145/360933.360975 Exclusion reasons: Q1–2 [II.ajk]Language exposition, theoretical discussion
478. A. A. diSessa & H. Abelson (1986): Boxer: a reconstructible computational medium. Communications of the ACM 29 (9). Pages 859-868. doi:10.1145/6592.6595 Exclusion reasons: Q1–2 [III.ajk]This article introduces a programming system. The language discussed in it is in essential respects graphical and thus misses our definition of a PL..
479. W. B. Dobrusky & T. B. Steel (1961): Universal computer-oriented language. Communications of the ACM 4 (3). Pages 138. doi:10.1145/366199.366220 Exclusion reasons: Q5 [III.ajk]This very brief report does not appear to describe empirical research.
480. Simon Dobson & Brian Matthews (2000): Ionic Types. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 296-312. doi:10.1007/3-540-45102-1_15 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
481. Mahesh Dodani & Chung-Shin Tsai (1992): ACTS: A type system for object-oriented programming based on abstract and concrete classes. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 308-328. doi:10.1007/BFb0053044 Exclusion reasons: Q5 [II.ajk]Formal type-theoretical work.
482. Jesse Doherty, Laurie Hendren & Soroush Radpour (2011): Kind analysis for MATLAB. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 99-118. doi:10.1145/2048066.2048077 Exclusion reasons: Q1–2 [III.ajk]This article deals with static analysis (for automatic program comprehension) of a single language; there is no issue regarding language design decisions.
483. Norihisa Doi, Yasushi Kodama & Ken Hirose (1988): An Implementation of an Operating System Kernel using Concurrent Object Oriented Language ABCL/c+. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 250-266. doi:10.1007/3-540-45910-3_15 Exclusion reasons: Q5 [III.ajk]This article describes a study in which an operating system is rewritten in another language, in order to show that the target language is capable of such use. This is clearly exploration of the language design's implications and thus is not empirical in our sense.
484. James Donahue & Alan Demers (1985): Data types are values. ACM Transactions on Programming Languages and Systems 7 (3). Pages 426-445. doi:10.1145/3916.3987 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
485. Christophe Dony (1988): An Object-oriented Exception Handling System for an Object-oriented Language. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 145-161. doi:10.1007/3-540-45910-3_9 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
486. Christophe Dony (1990): Exception handling and object-oriented programming: towards a synthesis. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 322-330. doi:10.1145/97945.97984 Exclusion reasons: Q5 [II.ajk]Feature development/exposition.
487. Christophe Dony, Jacques Malenfant & Pierre Cointe (1992): Prototype-based languages: from a new taxonomy to constructive proposals and their validation. In conference proceedings on Object-oriented programming systems, languages, and applications. New York, NY, USA: ACM. OOPSLA '92. Pages 201-217. doi:10.1145/141936.141954 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
488. Marko van Dooren & Eric Steegmans (2005): Combining the robustness of checked exceptions with the flexibility of unchecked exceptions using anchored exception declarations. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 455-471. doi:10.1145/1094811.1094847 Exclusion reasons: Q5 [II.ajk]Formal development.
489. Marko van Dooren & Eric Steegmans (2007): A Higher Abstraction Level Using First-Class Inheritance Relations. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 425-449. doi:10.1007/978-3-540-73589-2_20 Exclusion

reasons: Q5 [III.ajk]This article introduces, analyzes and evaluates a new set of language features. The evaluation is in the form of a "case study" (as the article calls it) in which Java programs are modified to use the new language features and the resulting code size changes are measured. As such, this evaluation is analytical even if it has the trappings of empiricity, as it explores the implications of the new technology, and not any contingent aspects of the world.

490. R. D. Dowsing & M. T. Sanderson (1986): Writing concurrent assemblers–a case study in path pascal. Software: Practice and Experience 16 (12). Pages 1117-1135. doi:10.1002/spe.4380161206 Exclusion reasons: Q1–2 [III.ajk]This article reports a study where a single algorithm is written several times, with different concurrency choices, in the same language (Path Pascal), and their performance is compared. There is no evaluation of a language design decision here.
491. J. R. Doyle & D. D. Stretch (1987): The classification of programming languages by usage. International Journal of Man-Machine Studies 26 (3). Pages 343-360. doi:10.1016/S0020-7373(87)80068-8 Exclusion reasons: Q1–2 [II.ajk]Counts language usage; no language design implications.
492. Derek Dreyer, Robert Harper, Manuel M. T. Chakravarty & Gabriele Keller (2007): Modular type classes. In Proc. 34th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 63-70. doi:10.1145/1190216.1190229 Exclusion reasons: Q5 [II.ajk]Elaboration and theoretical development of two language constructs, no empirical evaluation based on the abstract.
493. Jocelyn R. Drolet, Colin L. Moodie & Benoit Montreuil (1991): Object oriented simulation with Smalltalk-80: a case study. In Simulation Conference, 1991. Proceedings., Winter. Pages 312-322. doi:10.1109/WSC.1991.185629 Exclusion reasons: Q1–2 [III.ajk]This article presents a study in which Smalltalk was used in simulation. Although the article makes some claims about Smalltalk's efficacy, the study isn't designed to answer such questions (it's merely a "this is what we did" report).
494. Sophia Drossopoulou & Susan Eisenbach (1997): Java is type safe — Probably. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 389-418. doi:10.1007/BFb0053388 Exclusion reasons: Q5 [II.ajk]Type-theoretical work.
495. Sophia Drossopoulou, Ferruccio Damiani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2001): Fickle: Dynamic Object Re-classification. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 130-149. doi: 10.1007/3-540-45337-7_8 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
496. Sophia Drossopoulou, Ferruccio Damiani, Mariangiola Dezani-Ciancaglini & Paola Giannini (2002): More dynamic object reclassification: Fickle. ACM Transactions on Programming Languages and Systems 24 (2). Pages 153-191. doi:10.1145/514952.514955 Exclusion reasons: Q5 [II.ajk]Formal type-theoretical study.
497. Gilles Dubochet (2009): Computer Code as a Medium for Human Communication: Are Programming Languages Improving?. In PPIG 2009. (Found in http://ppig.org/workshops/21st-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article presents a laborarory experiment with human participants which compares, among other things, the program comprehensibility of dense and sparse style programming. While the styles do emphasize the use of different language constructs, those differences are not in the focus in this study and are not reported in any detail. Hence, this article cannot be considered to evaluate any language design decisions.
498. Stéphane Ducasse, Oscar Nierstrasz, Nathanael Schärli, Roel Wuyts & Andrew P. Black (2006): Traits: A mechanism for fine-grained reuse. ACM Transactions on Programming Languages and Systems 28 (2). Pages 331-388. doi:10.1145/1119479.1119483 Exclusion reasons: Q5 [III.ajk]This article is related to schärli-2003. Evaluation is analytical in this article as well.
499. R. Ducournau & M. Habib (1987): On Some Algorithms for Multiple Inheritance in Object Oriented Programming. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 243-252. doi:10.1007/3-540-47891-4_23 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
500. Roland Ducournau, Floréal Morandat & Jean Privat (2009): Empirical assessment of object-oriented implementations with multiple inheritance and static typing. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications. Pages 41-60. doi:10.1145/1640089.1640093 Exclusion reasons: Q1–2 [II.ajk]Comparison of implementation techniques.
501. Dominic Duggan (1999): Dynamic typing for distributed programming in polymorphic languages. ACM Transactions on Programming Languages and Systems 21 (1). Pages 11-45. doi:10.1145/314602.314604 Exclusion reasons: Q5 [III.ajk]This theoretical paper does not aspire to empiricity.
502. Dominic Duggan (2000): A Mixin-Based, Semantics-Based Approach to Reusing Domain-Specific Programming Languages. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 179-200. doi:10.1007/3-540-45102-1_9 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
503. Dominic Duggan & Jianhua Yao (2012): Static Sessional Dataflow. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 484-508. doi:10.1007/978-3-642-31057-7_22 Exclusion reasons: Q1–2 [II.ajk]No evaluation, at least based on the abstract.
504. Nan Dun & K. Taura (2012): An Empirical Performance Study of Chapel Programming Language. In Parallel and Distributed Processing Symposium Workshops PhD Forum (IPDPSW), 2012 IEEE 26th International. Pages 497-506. doi:10.1109/IPDPSW.2012.64 Exclusion reasons: Q1–2 [III.ajk]This study uses microbenchmarks and a couple of applications to compare Chapel to C, as measured by performance. This does not in any significant way evaluate the efficacy of any of the design decisions involved.
505. Fraser G. Duncan (1963): ECMA Subset of ALGOL 60. Communications of the ACM 6 (10). Pages 595-599. doi:10.1145/367651.1772994 Exclusion reasons: Q1–2 [III.ajk]This article is a brief language exposition.
506. Arthur G. Duncan (1982): Prototyping in ADA: a case study. In Proceedings of the workshop on Rapid prototyping. New York, NY, USA: ACM. Pages 54-60. doi:10.1145/1006259.1006269 Exclusion reasons: Q1–2 [II.ajk]No comparison.
507. H. E. Dunsmore & J. D. Gannon (1979): Data Referencing: An Empirical Investigation. Computer 12 (12). Pages 50-59. doi:10.1109/MC.1979.1658576 Exclusion reasons: Q1–2Q5 [III.ajk]This article reports an empirical study evaluating language design choices by empirical experimentation. There may also be an interesting lit review component. [posthoc] Initially included as study S40, but EXCLUDED post hoc. Although it looks like this study compares ST and NT, closer reading reveals the two languages are a red herring. There is no clear efficacy or PLDD issue.
508. H. E. Dunsmore & J. D. Gannon (1979): Analysis of the effects of programming factors on programming effort. Journal of Systems and Software 1. Pages 141-153. doi:10.1016/0164-1212(79)90014-1 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
509. Venkatreddy Dwarampudi, Shahbaz Singh Dhillon, Jivitesh Shah, Nikhil Joseph Sebastian & Nitin Kanigicharla (2010): Comparative study of the Pros and Cons of Programming languages Java, Scala, C++, Haskell, VB .NET, AspectJ, Perl, Ruby, PHP & Scheme - a Team 11 COMP6411-S10 Term Report. Paper in arXiv. http://arxiv.org/abs/1008.3431 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
510. Bjarne Däcker (2000): Concurrent Functional Programming for Telecommunications: A Case Study of Technology Introduction. . PhLic at Royal Institute of Technology, Stockholm. http://www.erlang.se/publications/bjarnelic.pdf Exclusion reasons: Q5 [III.ajk]This dissertation does not aspire to empiricity.
511. Françoise Détienne & Elliot Soloway (1990): An empirically-derived control structure for the process of program understanding. International Journal of Man-Machine Studies 33 (3). Pages 323-342. doi:10.1016/S0020-7373(05)80122-1 Exclusion reasons: Q1–2 [III.ajk]This article studies program comprehension.
512. Françoise Détienne (1997): Assessing the cognitive consequences of the object-oriented approach: A survey of empirical research on object-oriented design by individuals and teams. Interacting with Computers 9 (1). Pages 47-72. doi:10.1016/S0953-5438(97)00006-4 Exclusion reasons: Q1–2 [III.ajk]This article summarises and consolidates empirical research on object-oriented design, but does not focus on any language design decisions.
513. L. D'Amore, M. Guarracino, G. Laccetti & A. Murli (2004): Integrating Scientific Software Libraries in Problem Solving Environments: A Case Study with ScaLAPACK. Volume 3044.In Laganá, Antonio and Gavrilova, Marina and Kumar, Vipin and Mun, Youngsong and Tan, C. and Gervasi, Osvaldo (ed.) Computational Science and Its Applications – ICCSA 2004.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 515-524. doi:10.1007/978-3-540-24709-8_55 Exclusion reasons: Q1–2 [II.ajk]No language relevance.
514. M. Eaddy, T. Zimmermann, K.D. Sherwood, V. Garg, G.C. Murphy, N. Nagappan & A.V. Aho (2008): Do Crosscutting Concerns Cause Defects?. Software Engineering, IEEE Transactions on 34 (4). Pages 497-515. doi:10.1109/TSE.2008.36 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design issues.
515. Caroline M. Eastman (1982): A comment on English neologisms and programming language keywords. Communications of the ACM 25 (12). Pages 938-940. doi:10.1145/358728.358756 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
516. C.M. Eastman (1983): A lexical analysis of keywords in high level programming languages. International Journal of Man-Machine Studies 19 (6). Pages 595-607. doi:10.1016/S0020-7373(83)80073-X Exclusion reasons: Q1–2 [II.ajk]No clear efficacy issue.
517. K. Ebcioğlu, V. Saraswat & V. Sarkar (2005): X10: an Experimental Language for High Productivity Programming of Scalable Systems. In Second Workshop on Productivity and Performance in High-End Computing (PPHEC-05). Pages 45-52. http://www.research.ibm.com/arl/pphec/pphec2005-proceedings.pdf Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.
518. Alireza Ebrahimi (1994): Novice programmer errors: language constructs and plan composition. International Journal of Human-Computer Studies 41 (4). Pages 457-480. doi:10.1006/ijhc.1994.1069 Exclusion reasons: Q1–2 [III.ajk]This article studies error types, not language design issues.

519. Natalie Eckel & Joseph (Yossi) Gil (2000): Empirical Study of Object-Layout Strategies and Optimization Techniques. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 394-421. doi:10.1007/3-540-45102-1_20 Exclusion reasons: Q1–2 [II.ajk]Implementation techniques are studied.

520. Ernest Edmonds (1986): Negative knowledge toward a strategy for asking in logic programming. International Journal of Man-Machine Studies 24 (6). Pages 597-600. doi:10.1016/S0020-7373(86)80010-4 Exclusion reasons: Q5 [III.ajk]This short article only presents and illustrates a concept and does not evaluate its efficacy empirically.

521. Jonathan Edwards (2005): Subtext: uncovering the simplicity of programming. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 505-518. doi:10.1145/1094811.1094851 Exclusion reasons: Q5 [II.ajk]No indication of empirical work in abstract.

522. Jonathan Edwards (2007): No ifs, ands, or buts: uncovering the simplicity of conditionals. In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. Pages 639-658. doi:10.1145/1297027.1297075 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

523. W. Keith Edwards, Mark W. Newman, Jana Z. Sedivy & Trevor F. Smith (2009): Experiences with recombinant computing: Exploring ad hoc interoperability in evolving digital networks. ACM Transactions on Computer-Human Interaction 16 (1). Pages 3:1-3:44. doi:10.1145/1502800.1502803 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

524. Jonathan Edwards (2009): Coherent reaction. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 925-932. doi:10.1145/1639950.1640058 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

525. G. Efthivoulidis, N. Vlassis, P. Tsanakas & G. Papakonstantinou (1996): An experiment for truly parallel logic programming. Journal of Intelligent & Robotic Systems 16. Pages 169-184. doi:10.1007/BF00449704 Exclusion reasons: Q1–2 [III.ajk]This article evaluates implementation methods, not language design decisions.

526. B. Eichenauer, K. Kreuter, V. Haase, G. Müller & P. Holleczek (1973): PEARL, eine prozeß- und experimentorientierte Programmiersprache. Angewandte Informatik. Pages 363-372. http://md1.csa.com/partners/viewrecord.php?requester=gs&collection=TRD&recid=A7344388AH Exclusion reasons: Q4 [III.ajk]Article full text is in German.

527. Michael A. Eierman & Mark T. Dishaw (2007): The process of software maintenance: a comparison of object-oriented and third-generation development languages. Journal of Software Maintenance and Evolution: Research and Practice 19 (1). Pages 33-47. doi:10.1002/smr.343 Exclusion reasons: Q1–2 [III.ajk]This article reports a controlled experiment with human participants comparing COBOL and Smalltalk in maintenance. However, the dependent variables were all about perceived effort and thus the study does not speak about efficacy.

528. Marc Eisenstadt (1983): A user-friendly software environment for the novice programmer. Communications of the ACM 26 (11). Pages 1058-1064. doi:10.1145/358476.358500 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

529. Torbjörn Ekman & Görel Hedin (2004): Rewritable Reference Attributed Grammars. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 147-171. doi:10.1007/978-3-540-24851-4_7 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

530. P. V. Ellis (1962): An evaluation of autocode readability. Communications of the ACM 5 (3). Pages 156-159. doi:10.1145/366862.366888 Exclusion reasons: Q1–2 [III.ajk]This article does not repirt an evaluative study.

531. Burak Emir, Andrew Kennedy, Claudio Russo & Dachuan Yu (2006): Variance and Generalized Constraints for C# Generics. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 279-303. doi:10.1007/11785477_18 Exclusion reasons: Q5 [II.ajk]Type-theoretic study.

532. Burak Emir, Martin Odersky & John Williams (2007): Matching Objects with Patterns. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 273-298. doi:10.1007/978-3-540-73589-2_14 Exclusion reasons: Q1–2 [III.ajk]This article evaluates programming techniques empiricially for speed in one particular language and analytically in general. There was no PL design issue.

533. M. English, J. Buckley, T. Cahill & T. Lynch (2005): An empirical study of the use of friends in C++ software. In Program Comprehension, 2005. IWPC 2005. Proceedings. 13th International Workshop on. Pages 329-332. doi:10.1109/WPC.2005.7 Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study on the actual usage of friends in C++ code. It does not evaluate any language design decision for efficacy.

534. M. English, J. Buckley & T. Cahill (2005): A friend in need is a friend indeed . In Proceedings, 2005 International Symposium on Empirical Software Engineering, ISESE 2005. doi:10.1109/ISESE.2005.1541854 Exclusion reasons: Q1–2 [II.ajk]Metrics study, no PL design issue

535. Michael English, Jim Buckley & Tony Cahill (2010): A replicated and refined empirical study of the use of friends in C++ software. Journal of Systems and Software 83 (11). Pages 2275-2286. doi:10.1016/j.jss.2010.07.013 Exclusion reasons: Q1–2 [II.ajk]Study of usage patterns; no comparison language or construct.

536. M. C. Er (1984): On the complexity of recursion in problem-solving. International Journal of Man-Machine Studies 20 (6). Pages 537-544. doi:10.1016/S0020-7373(84)80028-0 Exclusion reasons: Q5 [III.ajk]This article is analytical in nature.

537. Sebastian Erdweg, Tillmann Rendel, Christian Kästner & Klaus Ostermann (2011): SugarJ: library-based syntactic language extensibility. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 391-406. doi:10.1145/2048066.2048099 Exclusion reasons: Q5 [III.ajk]The evaluation is analytical in nature.

538. Christoph Erhardt, Michael Stilkerich, Daniel Lohmann & Wolfgang Schröder-Preikschat (2011): Exploiting static application knowledge in a Java compiler for embedded systems: a case study. In Proceedings of the 9th International Workshop on Java Technologies for Real-Time and Embedded Systems. New York, NY, USA: ACM. Pages 96-105. doi:10.1145/2043910.2043927 Exclusion reasons: Q1–2 [II.ajk]This article appears to focus on an implementation technique.

539. George W. Ernst & William F. Ogden (1980): Specification of Abstract Data Types in Modula. ACM Transactions on Programming Languages and Systems 2 (4). Pages 522-543. doi:10.1145/357114.357117 Exclusion reasons: Q1–2 [II.ajk]No comparison.

540. Michael Ernst, Craig Kaplan & Craig Chambers (1998): Predicate dispatching: A unified theory of dispatch. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 186-211. doi:10.1007/BFb0054092 Exclusion reasons: Q1–2 [II.ajk]Theoretical work.

541. Erik Ernst (1999): Propagating Class and Method Combination. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 67-91. doi:10.1007/3-540-48743-3_4 Exclusion reasons: Q5 [III.ajk]This article presents a design and its implementation; there is no empirical work involved.

542. Erik Ernst (2001): Family Polymorphism. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 303-326. doi:10.1007/3-540-45337-7_17 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

543. M. D. Ernst, G. J. Badros & D. Notkin (2002): An empirical analysis of C preprocessor use. Software Engineering, IEEE Transactions on 28 (12). Pages 1146-1170. doi:10.1109/TSE.2002.1158288 Exclusion reasons: Q1–2 [II.ajk]Studies programming patterns; no PL design issue.

544. Erik Ernst (2003): Higher-Order Hierarchies. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 303-328. doi:10.1007/978-3-540-45070-2_14 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.

545. Patrick Th. Eugster, Rachid Guerraoui & Christian Heide Damm (2001): On objects and events. In OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 254-269 . doi:10.1145/504282.504301 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

546. Patrick Eugster (2007): Type-based publish/subscribe: Concepts and experiences. ACM Transactions on Programming Languages and Systems 29 (1). doi:10.1145/1180475.1180481 Exclusion reasons: Q5 [III.ajk]This article uses a running example to evaluate several design choices; but the evaluation is analytic in nature, not cotntaining any significant contingency.

547. Patrick Eugster & K. R. Jayaram (2009): EventJava: An Extension of Java for Event Correlation. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 570-594. doi:10.1007/978-3-642-03013-0_26 Exclusion reasons: Q1–2 [III.ajk]This article introduces, analyzes and evaluates a new Java extension. The empirical evaluation, however, is confined to a question of the reference implementation's performance and not to a question involving an actual PL design issue.

548. J. Ewer, B. Knight & D. Cowell (1995): Case study: an incremental approach to re-engineering a legacy FORTRAN Computational Fluid Dynamics code in C++. Advances in Engineering Software 22 (3). Pages 153 - 168. doi:10.1016/0965-9978(95)00021-N http://www.sciencedirect.com/science/article/pii/096599789500021N Exclusion reasons: Q1–2 [II.ajk]Focuses on converting from FORTRAN to C++; no language comparison so far as the abstract is concerned.

549. Richard A. Eyre-Todd (1993): The detection of dangling references in C++ programs. ACM Transactions on Programming Languages and Systems 2 (1-4). Pages 127-134. doi:10.1145/176454.176504 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

550. Dirk Fahland, Daniel Lübke, Jan Mendling, Hajo Reijers, Barbara Weber, Matthias Weidlich & Stefan Zugal (2009): Declarative versus Imperative Process

Modeling Languages: The Issue of Understandability. Volume 29.In Halpin, Terry and Krogstie, John and Nurcan, Selmin and Proper, Erik and Schmidt, Rainer and Soffer, Pnina and Ukor, Roland (ed.) Enterprise, Business-Process and Information Systems Modeling.Springer Berlin Heidelberg. Lecture Notes in Business Information Processing. Pages 353-366. doi:10.1007/978-3-642-01862-6_29 Exclusion reasons: Q1–2 [II.ajk]This article discusses modeling, not programming lagnuages.

551. Hans Fangohr (2004): A Comparison of C, MATLAB, and Python as Teaching Languages in Engineering. Volume 3039.In Bubak, Marian and van Albada, Geert and Sloot, Peter and Dongarra, Jack (ed.) Computational Science - ICCS 2004.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 1210-1217. doi:10.1007/978-3-540-25944-2_157 Exclusion reasons: Q5 [III.ajk]This article is analytical in nature and does not aspire to empiricity.

552. R. Fanta & V. Rajlich (1999): Restructuring legacy C code into C++. In Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on. Pages 77-85. doi:10.1109/ICSM.1999.792576 Exclusion reasons: Q1–2 [II.ajk]This article studies a program transformation, not any language design decisions.

553. Evan Farrer (2011): A Quantitative Analysis of Whether Unit Testing Obviates Static Type Checking for Error Detection. . Master's thesis. http://hdl.handle.net/10211.4/296 Exclusion reasons: Q1–2 [III.ajk]This master's thesis reports a study in which several existing, unit-tested Python programs were manually translated by the investigator into Haskell. In the process, several errors in each original program were discovered, the discovery being attributed to static type checking. There does not seem to be a clear language design issue at play, since as language design choices unit testing and static typing are not mutually exclusive.

554. Azadeh Farzan & Zachary Kincaid (2012): Verification of parameterized concurrent programs by modular reasoning about data and control. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 297-308. doi:10.1145/2103656.2103693 Exclusion reasons: Q1–2 [III.ajk]There's no language design issue here.

555. Richard J. Fateman (1982): High-Level Language Implications of the Proposed IEEE Floating-Point Standard. ACM Transactions on Programming Languages and Systems 4 (2). Pages 239-257. doi:10.1145/357162.357168 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

556. Leonidas Fegaras & Tim Sheard (1996): Revisiting catamorphisms over datatypes with embedded functions (or, programs from outer space). In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 284-294. doi:10.1145/237721.237792 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

557. Jerome A. Feldman & Paul D. Rovner (1969): An ALGOL-based associative language. Communications of the ACM 12 (8). Pages 439-449. doi:10.1145/363196.363204 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

558. Michael B. Feldman (1976): New languages from old: The extension of programming languages by embedding, with a case study. In Proceedings of the 2nd international conference on Software engineering. Los Alamitos, CA, USA: IEEE Computer Society Press. Pages 237-242. http://dl.acm.org/citation.cfm?id=800253.807682 Exclusion reasons: Q1–2 [II.ajk]Conceptual discussion and language exposition.

559. Mattias Felleisen & D. P. Friedman (1987): A calculus for assignments in higher-order languages. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 314-325. doi:10.1145/41625.41654 Exclusion reasons: Q5 [II.ajk]Formal development.

560. J. Fenton & K. Beck (1989): Playground: an object-oriented simulation system with agent rules for children of all ages. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 123-137. doi:10.1145/74877.74891 Exclusion reasons: Q3 [III.ajk]This article, in addition to the usual system exposition, reports briefly on an empirical study where schoolchildren used the system at school. However, the description of the study method is sketchy and important questions like the goal and data collection methods of the study are mostly left unanswered. I was not able to locate a more detailed report.

561. Sérgio Miguel Fernandes & João Cachopo (2011): Strict serializability is harmless: a new architecture for enterprise applications. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 257-276. doi:10.1145/2048147.2048221 Exclusion reasons: Q1–2 [III.ajk]There is no language design issue here.

562. Mary Fernández & Jérôme Siméon (2003): Growing XQuery. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 405-430. doi:10.1007/978-3-540-45070-2_18 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

563. Alan R. Feuer & Narain H. Gehani (1982): Comparison of the Programming Languages C and Pascal. ACM Computing Surveys 14 (1). Pages 73-92. doi:10.1145/356869.356872 Exclusion reasons: Q1–2 [III.ajk]This analytical paper does not aspire to empiricity.

564. Ludger Fiege, Mira Mezini, Gero Mühl & Alejandro P. Buchmann (2002): Engineering Event-Based Systems with Scopes. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 309-333. doi:10.1007/3-540-47993-7_14 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

565. John Field & Carlos A. Varela (2005): Transactors: a programming model for maintaining globally consistent distributed state in unreliable environments. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 195-208. doi:10.1145/1040305.1040322 Exclusion reasons: Q5 [II.ajk]Formal type-theoretic work.

566. E. Figueiredo, C. Sant'Anna, A. Garcia, T.T. Bartolomei, W. Cazzola & A. Marchetto (2008): On the Maintainability of Aspect-Oriented Software: A Concern-Oriented Measurement Framework. In Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on. Pages 183-192. doi:10.1109/CSMR.2008.4493313 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

567. E. Figueiredo, B. Silva, C. Sant'Anna, A. Garcia, J. Whittle & D. Nunes (2009): Crosscutting patterns and design stability: An exploratory analysis. In Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on. Pages 138-147. doi:10.1109/ICPC.2009.5090037 Exclusion reasons: Q1–2 [II.ajk]This article studies programming patterns, not language design issues.

568. Robert E. Filman (1987): Retrofitting objects. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 342-353. doi:10.1145/38765.38838 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

569. Robert Bruce Findler, Matthew Flatt & Matthias Felleisen (2004): Semantic Casts: Contracts and Structural Subtyping in a Nominal World. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 365-389. doi:10.1007/978-3-540-24851-4_17 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

570. G. R. Finnie (1986): Is top-down natural? : some experimental results from non-procedural languages. International Journal of Man-Machine Studies 25 (5). Pages 469-478. doi:10.1016/S0020-7373(86)80017-7 Exclusion reasons: Q1–2 [II.ajk]Studies development strategies, not language design issues.

571. Alice E. Fischer & Michael J. Fischer (1973): Mode modules as representations of domains: preliminary report. In Proc. 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 139-143. doi:10.1145/512927.512940 Exclusion reasons: Q1–2 [II.ajk]Feature exposition.

572. Jeffrey Fischer, Daniel Marino, Rupak Majumdar & Todd Millstein (2009): Fine-Grained Access Control with Object-Sensitive Roles. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 173-194. doi:10.1007/978-3-642-03013-0_9 Exclusion reasons: Q5 [III.ajk]The evaluation reported here is analytical in nature.

573. Kathleen Fisher & John Reppy (2000): Extending Moby with Inheritance-Based Subtyping. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 83-107. doi:10.1007/3-540-45102-1_5 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

574. Iztok Fister, Jr., Iztok Fister, Marjan Mernik & Janez Brest (2011): Design and implementation of domain-specific language easytime. Computer Languages, Systems & Structures 37 (4). Pages 151-167. doi:10.1016/j.cl.2011.04.001 Exclusion reasons: Q1–2 [III.ajk]This article does not compare its language to alternatives,

575. M. Fitter & T. R. G. Green (1979): When do diagrams make good computer languages?. International Journal of Man-Machine Studies 11 (2). Pages 235-261. doi:10.1016/S0020-7373(79)80019-X Exclusion reasons: Q1–2 [II.ajk]Diagram languages are not textual.

576. Matthew Flatt, Shriram Krishnamurthi & Matthias Felleisen (1998): Classes and mixins. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 171-183. doi:10.1145/268946.268961 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

577. Matthew Flatt (2002): Composable and compilable macros:: you want it when?. In Proceedings of the seventh ACM SIGPLAN international conference on Functional programming. New York, NY, USA: ACM. ICFP '02. Pages 72-83. doi:10.1145/581478.581486 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

578. A. C. Fleck (1990): A case study comparison of four declarative programming languages. Software: Practice and Experience 20 (1). Pages 49–65. doi:10.1002/spe.4380200107 Exclusion reasons: Q5 [III.ajk]This article compares four languages by programming the same algorithm in each. This is clearly an analytical study with no empirical ambitions.

579. Sebastian Fleissner & Elisa Baniassad (2008): Towards harmony-oriented programming. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 819-822. doi:10.1145/1449814.1449872 Exclusion reasons: Q5 [III.ajk]This discussion paper does not aspire to empiricity.

580. Sebastian Fleissner & Elisa Baniassad (2009): Harmony-oriented programming and software evolution. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 991-998. doi:10.1145/1639950.1640069 Exclusion reasons: Q5 [III.ajk]This article's evaluative work is analytical in nature.
581. Sebastian Fleissner & Elisa Baniassad (2009): Harmony-oriented smalltalk. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 747-748. doi:10.1145/1639950.1639995 Exclusion reasons: Q1–2 [III.ajk]This article is a brief system exposition.
582. Robert W. Floyd (1979): The paradigms of programming. Communications of the ACM 22 (8). Pages 455-460. doi:10.1145/359138.359140 Exclusion reasons: Q1–2 [III.ajk]This Turing award lecture, interesting as it is, does not report a study.
583. Brian Foote, Ralph E. Johnson & James Noble (2005): Efficient Multimethods in a Single Dispatch Language. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 337-361. doi:10.1007/11531142_15 Exclusion reasons: Q5 [III.ajk]This article does not empirically evaluate language design decisions (the performance measurements concern themselves merely with implementation speed).
584. Alfonso Caracciolo Di Forino (1963): Some remarks on the syntax of symbolic programming languages. Communications of the ACM 6 (8). Pages 456-460. doi:10.1145/366707.367584 Exclusion reasons: Q5 [III.ajk]This paper does not aspire to empiricity.
585. Ira R. Forman, Scott Danforth & Hari Madduri (1994): Composition of before/after metaclasses in SOM. In OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications. Pages 427-439. doi:10.1145/191080.191148 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
586. A. Forward, T.C. Lethbridge & D. Brestovansky (2009): Improving program comprehension by enhancing program constructs: An analysis of the Umple language. In Program Comprehension, 2009. ICPC '09. IEEE 17th International Conference on. Pages 311-312. doi:10.1109/ICPC.2009.5090073 Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.
587. C. C. Foster (1984): Cognitive strategies and looping constructs - an empirical study. Communications of the ACM 27 (10). Pages 1048-1048. (Missing from ACM Digital Library.) Exclusion reasons: Q1–2 [III.ajk]One-page letter to the editor.
588. Ian Foster & Stephen Taylor (1994): A compiler approach to scalable concurrent-program design. ACM Transactions on Programming Languages and Systems 16 (3). Pages 577-604. doi:10.1145/177492.177612 Exclusion reasons: Q1–2 [II.ajk]Implementation techniques
589. Ian Foster (1996): Compositional parallel programming languages. ACM Transactions on Programming Languages and Systems 18 (4). Pages 454-476. doi:10.1145/233561.233565 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
590. J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce & Alan Schmitt (2005): Combinators for bi-directional tree transformations: a linguistic approach to the view update problem. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 233-246. doi:10.1145/1040305.1040325 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
591. J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce & Alan Schmitt (2007): Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Transactions on Programming Languages and Systems 29 (3). doi:10.1145/1232420.1232424 Exclusion reasons: Q5 [III.ajk]This extensive analytic-constructive paper does not aspire to empiricity.
592. Mathieu Fourment & Michael R Gillings (2008): A comparison of common programming languages used in bioinformatics. BMC Bioinformatics 9. Pages 82. doi:10.1186/1471-2105-9-82 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which a single programmer implemented the same algorithms in several languages and their run-time performance using particular language implementations and some source code metrics were compared. Since the comparison is multiway, there is no clear language design decision at play; to the extent there might be, it's not evaluated empirically.
593. Cédric Fournet & Georges Gonthier (1996): The reflexive CHAM and the join-calculus. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 372-385. doi:10.1145/237721.237805 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
594. Cédric Fournet & Tamara Rezk (2008): Cryptographically sound implementations for typed information-flow security. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 323-335. doi:10.1145/1328438.1328478 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
595. Pascal Fradet & Daniel Le Métayer (1997): Shape types. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 27-39. doi:10.1145/263699.263706 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
596. Nissim Francez & Shaula A. Yemini (1985): Symmetric intertask communication. ACM Transactions on Programming Languages and Systems 7 (4). Pages 622-636. doi:10.1145/4472.4475 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
597. Christopher W. Fraser & David R. Hanson (1985): High-level language facilities for low-level services. In Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 217-224. doi:10.1145/318593.318642 Exclusion reasons: Q1–2 [II.ajk]Language and system exposition.
598. Thomas P. Frazier, John W. Bailey & Melissa L. Corso (1996): Comparing ada and FORTRAN lines of code: Some experimental results. Empirical Software Engineering 1 (1). Pages 45-59. doi:10.1007/BF00125811 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which sevearl standard FORTRAN programs were rewritten in both FORTRAN and Ada, in oorder to determine the relationship between lines of FORTRAN code and lines of Ada code for the same functionality. This study does not have any programming language design relevance so far as I can tell.
599. S. M. G. Freeman & M. S. Manasse (1994): Adding digital video to an object-oriented user interface toolkit. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 493-512. doi:10.1007/BFb0052198 Exclusion reasons: Q1–2 [II.ajk]No language comparison.
600. Steve Freeman & Nat Pryce (2006): Evolving an embedded domain-specific language in Java. In OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. Pages 855-865. doi:10.1145/1176617.1176735 Exclusion reasons: Q5 [III.ajk]This article, so far as it has any empirical content, is an experience report and is thus excluded under our protocol.
601. Bjorn N. Freeman-Benson & Alan Borning (1992): Integrating constraints with an object-oriented language. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 268-286. doi:10.1007/BFb0053042 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper has no aspiration to empiricity.
602. Daniel P. Friedman & David S. Wise (1978): A note on conditional expressions. Communications of the ACM 21 (11). Pages 931-933. doi:10.1145/359642.359650 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
603. Daniel P. Friedman & David S. Wise (1980): An indeterminate constructor for applicative programming. In Proc. 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 245-250. doi:10.1145/567446.567470 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
604. Daniel P. Friedman & Christopher T. Haynes (1985): Constraining control. In Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 245-254. doi:10.1145/318593.318654 Exclusion reasons: Q5 [III.ajk]This article introduces a new construct. It has no aspiration for empiricity.
605. Svend Frølund (1992): Inheritance of synchronization constraints in concurrent object-oriented programming languages. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 185-196. doi:10.1007/BFb0053037 Exclusion reasons: Q5 [III.ajk]This analytical study does not aspire to empiricity.
606. Svend Frølund & Gul Agha (1993): A Language Framework for Multi-Object Coordination. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 346-360. doi:10.1007/3-540-47910-4_18 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
607. Koichi Fukunaga & Shin-ichi Hirose (1986): An experience with a Prolog-based object-oriented language. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 224-231. doi:10.1145/28697.28719 Exclusion reasons: Q1–2 [II.ajk]No comparison.
608. Michael Furr & Jeffrey S. Foster (2008): Checking type safety of foreign function calls. ACM Transactions on Programming Languages and Systems 30 (4). doi:10.1145/1377492.1377493 Exclusion reasons: Q1–2 [II.ajk]Study of program analysis.
609. Kokichi Futatsugi, Joseph A. Goguen, Jean-Pierre Jouannaud & José Meseguer (1985): Principles of OBJ2. In Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 52-66. doi:10.1145/318593.318610 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
610. R. Fyfe (1997): An Empirical Study on C++ Programs Project Proposal. Student project report. (No URL available at this time.) Exclusion reasons: Q3 [III.ajk]Considering fyfe-1997a and fyfe-1997b, this is likely a document related to the same study. As such, its disposition does not really matter.
611. Bent Gabelgaard (1992): Using object-oriented programming techniques for implementing ISDN supplementary services. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 213-232. doi:10.1007/BFb0053039 Exclusion reasons:

Q5 [III.ajk]The paper's Section 3 claims to support the contention that "the use of an object-oriented implementation language like BETA will - in comparison with conventional languages - give the ISDN implementor a better tool for implementing ISDN supplementary services by supporting a natural separation between the BCP code and the code for the supplementary services". However, it does not. It is a conventional analytical (or perhaps constructive) presentation arguably demonstrating that BETA can (as opposed to "will") function in the claimed manner. The rest of the paper does not aspire even that much to empiricity.

612. Pedro Gabriel, Miguel Goulão & Vasco Amaral (2011): Do Software Languages Engineers Evaluate their Languages?. Paper in arXiv. http://arxiv.org/abs/1109.6794 Exclusion reasons: Q7 [III.ajk]This article does not discuss empirical evidence in any detail.
613. B. A. Galler & A. J. Perlis (1966): A proposal for definitions in ALGOL. Communications of the ACM 9 (7). Pages 481-482. doi:10.1145/365719.366429 Exclusion reasons: Q5 [III.ajk]This abstract does not aspire to empiricity.
614. B. A. Galler & A. J. Perlis (1967): A proposal for definitions in ALGOL. Communications of the ACM 10 (4). Pages 204-219. doi:10.1145/363242.363252 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
615. Kathleen M. Galotti & William F. Ganong III (1985): What non-programmers know about programming: Natural language procedure specification. International Journal of Man-Machine Studies 22 (1). Pages 1-10. doi:10.1016/S0020-7373(85)80073-0 Exclusion reasons: Q1–2 [II.ajk]The basic issue in this article seems to be whether and how people not trained to program use control-flow constructs analoguous to those used in programming. This does not reflect in any way the efficacy of such constructs in the hands of trained programmers, whether novice or expert.
616. Erich Gamma, André Weinand & Rudolf Marty (1989): Integration of a Programming Environment into ET++ - A Case Study. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 283-294. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890283.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.
617. W.M Gancza & T Paszkiewicz (1998): The application of object-oriented programming to Monte Carlo experiments on beams of phonons in crystals. Computers & Chemistry 22 (1). Pages 21-30. doi:10.1016/S0097-8485(97)00012-0 Exclusion reasons: Q1–2 [II.ajk]No PL design issue; no comparison language.
618. J. D. Gannon (1978): Characteristic errors in programming languages. In Proceedings of the 1978 annual conference - Volume 2. New York, NY, USA: ACM. ACM '78. Pages 570-575. doi:10.1145/800178.810093 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
619. John Gannon, Paul McMullin & Richard Hamlet (1981): Data Abstraction, Implementation, Specification, and Testing. ACM Transactions on Programming Languages and Systems 3 (3). Pages 211-223. doi:10.1145/357139.357140 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
620. J. D. Gannon, E. E. Katz & V. R. Basili (1986): Metrics for Ada packages: an initial study. Communications of the ACM 29 (7). Pages 616-623. doi:10.1145/6138.6144 Exclusion reasons: Q1–2 [II.ajk]Evaluation of a single language.
621. Vladimir Gapeyev & Benjamin C. Pierce (2003): Regular Object Types. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 151-175. doi:10.1007/978-3-540-45070-2_8 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work.
622. Alessandro F Garcia, Cecília M.F Rubira, Alexander Romanovsky & Jie Xu (2001): A comparative study of exception handling mechanisms for building dependable object-oriented software. Journal of Systems and Software 59 (2). Pages 197-222. doi:10.1016/S0164-1212(01)00062-0 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
623. Ronald Garcia, Jaakko Jarvi, Andrew Lumsdaine, Jeremy G. Siek & Jeremiah Willcock (2003): A comparative study of language support for generic programming. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. Pages 115-134. doi:10.1145/949305.949317 Exclusion reasons: Q5 [III.ajk]This article reports an analytical comparison of the generic programming facilities of several programming languages. There is no empiricity involved.
624. Alessandro Garcia, Cláudio Sant'Anna, Christina Chavez, Viviane da Silva, Carlos de Lucena & Arndt von Staa (2004): Separation of Concerns in Multi-agent Systems: An Empirical Study. Volume 2940.In Lucena, Carlos and Garcia, Alessandro and Romanovsky, Alexander and Castro, Jaelson and Alencar, Paulo (ed.) Software Engineering for Multi-Agent Systems II.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 343-344. doi:10.1007/978-3-540-24625-1_4 Exclusion reasons: Q1–2 [III.ajk]This article studies development approaches, not any language design decisions.
625. Alessandro Garcia, Cláudio Sant'Anna, Eduardo Figueiredo, Uirá Kulesza, Carlos Lucena & Arndt von Staa (2005): Modularizing design patterns with aspects: a quantitative study. In Proceedings of the 4th international conference on Aspect-oriented software development. New York, NY, USA: ACM. AOSD '05. Pages 3-14. doi:10.1145/1052898.1052899 Exclusion reasons: Q1–2 [III.ajk]This article studies the interaction of design patterns an aspect orientation. It does not present an efficacy question from a programmer's point of view.
626. D. Garlan, R. Allen & J. Ockerbloom (1995): Architectural mismatch: why reuse is so hard. Software, IEEE 12 (6). Pages 17-26. doi:10.1109/52.469757 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
627. L. Nancy Garrett & Karen E. Smith (1986): Building a timeline editor from prefab parts: the architecture of an object-oriented application. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 202-213. doi:10.1145/28697.28717 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a PL design decision.
628. H. Garsden & A. L. Wendelborn (1992): Experiments with pipelining parallelism in SISAL. Volume ii.In System Sciences, 1992. Proceedings of the Twenty-Fifth Hawaii International Conference on. Pages 251-262. doi:10.1109/HICSS.1992.183297 Exclusion reasons: Q1–2 [III.ajk]This article advocates a particular language and a particular manner of writing programs in it. There is no PL design issue here.
629. Jan V. Garwick (1964): GARGOYLE: a language for compiler writing. Communications of the ACM 7 (1). Pages 16-20. doi:10.1145/363872.363894 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
630. Jan V. Garwick (1968): Programming Languages: GPL, a truly general purpose language. Communications of the ACM 11 (9). Pages 634-638. doi:10.1145/364063.364092 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
631. Andreas Gawecki & Florian Matthes (1996): Integrating subtyping, matching and type quantification: A practical perspective. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 26-47. doi:10.1007/BFb0053055 Exclusion reasons: Q5 [III.ajk]This article does not empirically evaluate anything.
632. H. J. Gawlik (1963): MIRFAC: a compiler based on standard mathematical notation and plain English. Communications of the ACM 6 (9). Pages 545-547. doi:10.1145/367593.367618 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
633. David Gelernter (1985): Generative communication in Linda. ACM Transactions on Programming Languages and Systems 7 (1). Pages 80-112. doi:10.1145/2363.2433 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
634. D. Gelernter, S. Jagannathan & T. London (1987): Environments as first class objects. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 98-110. doi:10.1145/41625.41634 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
635. Michael R. Genesereth & Matthew L. Ginsberg (1985): Logic programming. Communications of the ACM 28 (9). Pages 933-941. doi:10.1145/4284.4287 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
636. Charles M. Geschke, Jr. James H. Morris & Edwin H. Satterthwaite (1977): Early experience with Mesa. Communications of the ACM 20 (8). Pages 540-553. doi:10.1145/359763.359771 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
637. Henda Hadjami Ben Ghezala & Farouk Kamoun (1995): A reuse approach based on object orientation: its contributions in the development of CASE tools. In Proceedings of the 1995 Symposium on Software reusability. New York, NY, USA: ACM. SSR '95. Pages 53-62. doi:10.1145/211782.211798 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
638. Carlo Ghezzi, Matteo Pradella & Guido Salvaneschi (2010): Programming language support to context-aware adaptation: a case-study with Erlang. In Proceedings of the 2010 ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems. New York, NY, USA: ACM. Pages 59-68. doi:10.1145/1808984.1808991 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
639. Celina Gibbs, Chunjian Robin Liu & Yvonne Coady (2005): Sustainable System Infrastructure and Big Bang Evolution: Can Aspects Keep Pace?. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 241-261. doi:10.1007/11531142_11 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.
640. Joseph (Yossi) Gil & Alon Itai (1998): The complexity of type analysis of object oriented programs. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 601-634. doi:10.1007/BFb0054109 Exclusion reasons: Q1–2 [II.ajk]Type analysis algorithm theory.
641. Joseph (Yossi) Gil & Yuval Shimron (2011): Smaller footprint for Java collections. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 191-192. doi:10.1145/2048147.2048201 Exclusion reasons: Q1–2 [II.ajk]This article seems to evaluate implementation techniques only.
642. Joseph Gil & Yuval Shimron (2012): Smaller Footprint for Java Collections. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 356-382. doi:10.1007/978-3-642-31057-7_17 Exclusion reasons: Q1–2 [II.ajk]This article discusses either implementation or programming techniques only, not sure which.

643. D. J. Gilmore & H. T. Smith (1984): An investigation of the utility of flowcharts during computer program debugging. International Journal of Man-Machine Studies 20 (4). Pages 357-372. doi:10.1016/S0020-7373(84)80074-7 Exclusion reasons: Q1–2 [II.ajk]Flowcharts are not a PL by our definition.

644. David Gilmore (1991): Visbility - A New Dimension?. In PPIG 1991. (Found in http://ppig.org/workshops/3rd-programme.html.) Exclusion reasons: Q1–2 [III.ajk]Decision based on a later version of the paper, published in People and Computers VI (CUP 1991). The empirical study reported does not evaluate a language design decision, nor is it directly relevant to PL design in general.

645. David Gilmore (1994): Ceilidh. In PPIG 1994. (Found in http://ppig.org/workshops/6th-programme.html.) Exclusion reasons: Q3 [III.ajk]The interlibrary loan service was not able to locate a copy; they even asked the author. See msgid <04E17A7BB138C642A2B06A667D96AA1E256CBDB9@mbs1.ad.jyu.fi> in misc/interlibrary-service-rejects.mbox

646. James F. Gimpel (1972): Blocks - a new datatype for SNOBOL4. Communications of the ACM 15 (6). Pages 438-447. doi:10.1145/361405.361410 Exclusion reasons: Q5 [III.ajk]This article is a language feature exposition, with no attempt at empiricity.

647. J. F. Gimpel (1973): A theory of discrete patterns and their implementation in SNOBOL4. Communications of the ACM 16 (2). Pages 91-100. doi: 10.1145/361952.361960 Exclusion reasons: Q1–2 [II.ajk]Theoretical and implementation study.

648. Giuseppina C. Gini & M. L. Gini (1985): Dealing with world-model-based programs. ACM Transactions on Programming Languages and Systems 7 (2). Pages 334-347. doi:10.1145/3318.3479 Exclusion reasons: Q1–2 [II.ajk]Language exposition at best.

649. DANIELA GIORDANO (2002): Evolution of interactive graphical representations into a design language: a distributed cognition account. International Journal of Human-Computer Studies 57 (4). Pages 317-345. doi:10.1006/ijhc.2002.1020 Exclusion reasons: Q1–2 [III.ajk]Graphical languages are excluded.

650. J. Glauert, J. Kennaway & M. Sleep (1991): Dactl: An experimental graph rewriting language. Volume 532.In Ehrig, Hartmut and Kreowski, Hans-Jörg and Rozenberg, Grzegorz (ed.) Graph Grammars and Their Application to Computer Science. Lecture Notes in Computer Science. Pages 378-395. doi:10.1007/BFb0017401 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

651. Ephraim P. Glinert (1989): Towards software metrics for visual programming. International Journal of Man-Machine Studies 30 (4). Pages 425-445. doi:10.1016/S0020-7373(89)80026-4 Exclusion reasons: Q1–2 [II.ajk]Visual programming is excluded.

652. Guy Golan-Gueta, Nathan Bronson, Alex Aiken, G. Ramalingam, Mooly Sagiv & Eran Yahav (2011): Automatic fine-grain locking using shape properties. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 225-242. doi:10.1145/2048066.2048086 Exclusion reasons: Q1–2 [III.ajk]There is no language design issue here.

653. David Goldberg (1992): The design of floating-point data types. ACM Transactions on Programming Languages and Systems 1 (2). Pages 138-151. doi:10.1145/151333.151373 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

654. David S. Goldberg, Robert Bruce Findler & Matthew Flatt (2004): Super and inner: together at last!. In OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 116-129. doi:10.1145/1028976.1028987 Exclusion reasons: Q5 [III.ajk]This article motivates and introduces a construct and its implementation. It has no empirical ambition.

655. Roy Goldfinger (1961): Problem-oriented programming language structure. Communications of the ACM 4 (3). Pages 138. doi:10.1145/366199.366222 Exclusion reasons: Q1–2 [III.ajk]This short note describes a research project and does not report a study.

656. D. Goldson (2004): An experiment in the design of distributed programs. In Software Engineering Conference, 2004. Proceedings. 2004 Australian. Pages 70-76. doi:10.1109/ASWEC.2004.1290459 Exclusion reasons: Q5 [III.ajk]This paper has no aspiration to empiricity.

657. Fernando Gomez & Viva Wingate (1991): TQ: a language for specifying programming problems. International Journal of Man-Machine Studies 35 (5). Pages 633-656. doi:10.1016/S0020-7373(05)80181-6 Exclusion reasons: Q1–2 [II.ajk]Specification language is not a programming language, very likely.

658. Michael D. Good, John A. Whiteside, Dennis R. Wixon & Sandra J. Jones (1984): Building a user-derived interface. Communications of the ACM 27 (10). Pages 1032-1043. doi:10.1145/358274.358284 Exclusion reasons: Q1–2 [II.ajk]Evaluates a design approach, not a design decision.

659. J Good (1996): The'Right'Tool for the Task: an Investigation of External Representations, Program Abstractions and Task Requirements. In Proc. Empirical Studies of Programmers 1996. Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any (textual) programming language design decisions.

660. Judith Good (1999): Getting a GRiP on the comprehension of data-flow visual programming languages. In PPIG 1999. (Found in http://ppig.org/workshops/11th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This short article deals with visual languages, which we have excluded in the protocol.

661. Judith Good & Paul Brna (2004): Program comprehension and authentic measurement:: a scheme for analysing descriptions of programs. International Journal of Human-Computer Studies 61 (2). Pages 169-185. doi:10.1016/j.ijhcs.2003.12.010 Exclusion reasons: Q1–2 [II.ajk]No language design issue.

662. John B. Goodenough (1975): Structured exception handling. In Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 204-224. doi:10.1145/512976.512997 Exclusion reasons: Q5 [III.ajk]This article discusses analytically the programming language design issue of exception handling. There is no aspiration to empiricality.

663. John B. Goodenough (1975): Exception handling design issues. SIGPLAN Notices 10 (7). Pages 41-45. doi:10.1145/987305.987313 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

664. John B. Goodenough (1975): Exception handling: issues and a proposed notation. Communications of the ACM 18 (12). Pages 683-696. doi:10.1145/361227.361230 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

665. Leonard Goodwin & Mohammad Sanati (1986): Learning computer programming through dynamic representation of computer functioning: evaluation of a new learning package for Pascal. International Journal of Man-Machine Studies 25 (3). Pages 327-341. doi:10.1016/S0020-7373(86)80064-5 Exclusion reasons: Q1–2 [II.ajk]This article evaluates a teaching aid.

666. M. Gordon, R. Milner, L. Morris, M. Newey & C. Wadsworth (1978): A Metalanguage for interactive proof in LCF. In Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 119-130. doi:10.1145/512760.512773 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.

667. Colin S. Gordon, Matthew J. Parkinson, Jared Parsons, Aleks Bromfield & Joe Duffy (2012): Uniqueness and reference immutability for safe parallelism. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 21-40. doi:10.1145/2384616.2384619 Exclusion reasons: Q3 [III.ajk]Note the extended technical report at http://research.microsoft.com/apps/pubs/default.aspx?id=170528 . The only arguably empirical part of the study is where the authors describe the experiences of a Microsoft developer team. The methodology for collecting these experiences is not described at all.

668. Sergei Gorlatch (2004): Send-receive considered harmful: Myths and realities of message passing. ACM Transactions on Programming Languages and Systems 26 (1). Pages 47-56. doi:10.1145/963778.963780 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

669. S. Gorn (1964): Report on Input-Output Procedures for ALGOL 60. Communications of the ACM 7 (10). Pages 628-630. doi:10.1145/364888.876697 Exclusion reasons: Q1–2 [III.ajk]This is a language fragment exposition.

670. S. Gorn (1964): FORTRAN vs. Basic FORTRAN: a programming language for informational processing on automatic data processing systems. Communications of the ACM 7 (10). Pages 591-624. doi:10.1145/364888.876694 Exclusion reasons: Q1–2 [III.ajk]This article is a side-by-side juxtaposition of two similar langauge definitions, with no other content.

671. S. Gorn (1964): Report on SUBSET ALGOL 60 (IFIP). Communications of the ACM 7 (10). Pages 626-628. doi:10.1145/364888.876696 Exclusion reasons: Q1–2 [III.ajk]This language specification does not report a study.

672. Herkimer J. Gottfried & Margaret M. Burnett (1997): Programming complex objects in spreadsheets: an empirical study comparing textual formula entry with direct manipulation and gestures. In Papers presented at the seventh workshop on Empirical studies of programmers. Pages 42-68. doi:10.1145/266399.266405 Exclusion reasons: Q1–2 [II.ajk]Based on the title, compares a textual format to nontextual. Under this study's definition of a programming language, the latter is easily disqualified. Thus, there is no comparison in this study, even if one were to assume that this is PL relevant.

673. John D. Gould (1975): Some psychological evidence on how people debug computer programs. International Journal of Man-Machine Studies 7 (2). Pages 151-182. doi:10.1016/S0020-7373(75)80005-8 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate language design decisions.

674. Irene Govender (2008): Solving introductory computer programming problems using an object-oriented language : insights from an empirical study. African Journal of Research in Mathematics, Science and Technology Education 12. Pages 19-33. http://www.sabinet.co.za/abstracts/saarmste/saarmste_v12_si1_a3.html Exclusion reasons: Q1–2 [II.ajk]Study of programmer behaviour, no PL design issue.

675. Irene Govender (2010): From Procedural to Object-Oriented Programming (OOP): An exploratory study of teachers' performance. South African Computer Journal 46. http://sacj.cs.uct.ac.za/index.php/sacj/article/viewArticle/13 Exclusion reasons: Q1–2 [III.ajk]This article studies a matter of teaching object-oriented programming. There is no PL design issue here.

676. Marvin Lowell Graham & Peter Zilahy Ingerman (1965): An assembly language for reprogramming. Communications of the ACM 8 (12). Pages 769-772. doi:10.1145/365691.365935 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

677. Kathryn E. Gray (2008): Safe Cross-Language Inheritance. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes

in Computer Science 5142. Pages 52-75. doi:10.1007/978-3-540-70592-5_4 Exclusion reasons: Q5 [II.ajk]Formal type theoretical work.

678. Julien Green (1959): Possible modification to the international algebraic language. Communications of the ACM 2 (2). Pages 6-8. doi:10.1145/368280.368285 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
679. Julien Green (1959): Remarks on ALGOL and symbol manipulation. Communications of the ACM 2 (9). Pages 25-27. doi:10.1145/368424.368438 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
680. Julien Green (1966): Microprogramming, emulators and programming languages. Communications of the ACM 9 (3). Pages 230-232. doi:10.1145/365230.365276 Exclusion reasons: Q5 [III.ajk]This paper has no empirical aspirations.
681. T. R. G. Green & D. J. Guest (1974): An Easily-implemented Language for Computer Control of Complex Experiments. International Journal of Man-Machine Studies 6 (3). Pages 335-359. doi:10.1016/S0020-7373(74)80026-X Exclusion reasons: Q1–2 [II.ajk]Language exposition, not an evaluative study.
682. T. R. G. Green & S. J. Payne (1984): Organization and learnability in computer languages. International Journal of Man-Machine Studies 21 (1). Pages 7-18. doi:10.1016/S0020-7373(84)80035-8 Exclusion reasons: Q1–2 [III.ajk]There is no programming language issue here.
683. Thomas RG Green, Marian Petre & RKE Bellamy (1991): Comprehensibility of visual and textual programs: A test of superlativism against the'match-mismatch'conjecture. Volume 121146.In Empirical studies of programmers: Fourth workshop. Norwood, NJ: Ablex. Exclusion reasons: Q1–2 [II.ajk]This article compares a graphical and a textual language. Since only textual languages are admissible in this study, there is no real comparison and thus no admissible design decision under study.
684. T. R. G. Green & M. Petre (1992): When Visual Programs are Harder to Read than Textual Programs. In Human-Computer Interaction: Tasks and Organisation, Proc. ECCE-6 (6th European Conference on Cognitive Ergonomics). http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.57.1633 Exclusion reasons: Q1–2 [III.ajk]This article compares visual and textual languages. Since we admit only textual languages here, there is no admissible language design decision at play.
685. T. R. G. Green, M. M. Burnett, A. J. Ko, K. J. Rothermel, C. R. Cook & J. Schonfeld (2000): Using the cognitive walkthrough to improve the design of a visual programming experiment. In Visual Languages, 2000. Proceedings. 2000 IEEE International Symposium on. Pages 172-179. doi:10.1109/VL.2000.874381 Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded per our definition of PL.
686. Thomas R. G. Green (2002): Arenas of Interest in Designing a Notation: How Far do Cognitive Dimensions Go?. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q3 [III.ajk]Appears to have been an unpublished lecture.
687. Steven Greene, Roger Ratcliff & Gail McKoon (1988): A flexible programming language for generating stimulus lists for cognitive psychology experiments. Behavior Research Methods 20 (2). Pages 119-128. doi:10.3758/BF03203815 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
688. Sharon L. Greene, Susan J. Devlin, Philip E. Cannata & Louis M. Gomez (1990): No IFs, ANDs, or ORs: A study of database querying. International Journal of Man-Machine Studies 32 (3). Pages 303-326. doi:10.1016/S0020-7373(08)80005-3 Exclusion reasons: Q1–2 [II.ajk]No programming languages here.
689. Aaron Greenhouse & John Boyland (1999): An Object-Oriented Effects System. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 205-229. doi:10.1007/3-540-48743-3_10 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
690. Douglas Gregor, Jaakko Järvi, Jeremy Siek, Bjarne Stroustrup, Gabriel Dos Reis & Andrew Lumsdaine (2006): Concepts: linguistic support for generic programming in C++. In OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. Pages 291-310. doi:10.1145/1167473.1167499 Exclusion reasons: Q1–2 [II.ajk]Language feature exposition.
691. Saso Greiner, Damijan Rebernak, Janez Brest & Viljem Zumer (2005): Z0 - a tiny experimental language. SIGPLAN Notices 40 (8). Pages 19-28. doi:10.1145/1089851.1089856 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
692. Clemens Grelck & Sven-Bodo Scholz (2000): HPF vs. SAC — A Case Study. Volume 1900.In Bode, Arndt and Ludwig, Thomas and Karl, Wolfgang and Wismüller, Roland (ed.) Euro-Par 2000 Parallel Processing.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 620-624. doi:10.1007/3-540-44520-X_87 Exclusion reasons: Q5 [III.ajk]The arguably empirical content of this paper is really evaluating implementations, not language design decisions.
693. Lee L. Gremillion (1984): Determinants of program repair maintenance requirements. Communications of the ACM 27 (8). Pages 826-832. doi:10.1145/358198.358228 Exclusion reasons: Q1–2 [II.ajk]No programming language relevance.
694. Marek Greniewski & Wladyslaw Turski (1963): The external language KLIPA for the URAL-2 digital computer. Communications of the ACM 6 (6). Pages 321-324. doi:10.1145/366604.366654 Exclusion reasons: Q5 [III.ajk]This language exposition does not report an evaluative study.
695. David Gries & Narain Gehani (1977): Some ideas on data types in high-level languages. Communications of the ACM 20 (6). Pages 414-420. doi:10.1145/359605.359624 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
696. Ralph E. Griswold & David R. Hanson (1980): An Alternative to the Use of Patterns in String Processing. ACM Transactions on Programming Languages and Systems 2 (2). Pages 153-172. doi:10.1145/357094.357096 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
697. Ralph E. Griswold, David R. Hanson & John T. Korb (1981): Generators in Icon. ACM Transactions on Programming Languages and Systems 3 (2). Pages 144-161. doi:10.1145/357133.357136 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
698. Peter Grogono & Markku Sakkinen (2000): Copying and Comparing: Problems and Solutions. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 226-250. doi:10.1007/3-540-45102-1_11 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
699. Paul Gross & Kris Powers (2005): Evaluating assessments of novice programming environments. In Proceedings of the first international workshop on Computing education research. New York, NY, USA: ACM. ICER '05. Pages 99-110. doi:10.1145/1089786.1089796 Exclusion reasons: Q1–2 [III.ajk]This article summarises and evaluates studies of programming environments. Their relevance to textual language design decisions is too far removed to count here.
700. Dan Grossman (2006): Quantified types in an imperative language. ACM Transactions on Programming Languages and Systems 28 (3). Pages 429-475. doi:10.1145/1133651.1133653 Exclusion reasons: Q5 [II.ajk]Formal type-theoretic treatment.
701. Christian Grothoff, Jens Palsberg & Jan Vitek (2007): Encapsulating objects with confined types. ACM Transactions on Programming Languages and Systems 29 (6). doi:10.1145/1286821.1286823 Exclusion reasons: Q1–2 [II.ajk]Program analysis tool exposition.
702. Thomas Gruber (1987): XREF: a case study in Common Lisp portability. SIGPLAN Lisp Pointers 1 (1). Pages 5-12. doi:10.1145/1862396.1862397 Exclusion reasons: Q1–2 [III.ajk]This article does not compare Common Lisp to anything else and thus does not evaluate any of the design decisions it embodies.
703. V. Gruhn & R. Laue (2007): On Experiments for Measuring Cognitive Weights for Software Control Structures. In Cognitive Informatics, 6th IEEE International Conference on. Pages 116-119. doi:10.1109/COGINF.2007.4341880 Exclusion reasons: Q1–2 [II.ajk]Metrics work, no PL design issue.
704. Kaj Grønbæk (1989): Rapid Prototyping With Fourth Generation Systems–An Empirical Study. Information Technology & People 5 (2). Pages 105–125. doi:10.1108/EUM0000000003530 Exclusion reasons: Q1–2 [II.ajk]No language design issue.
705. Sharath Chowdary Gude (2012): JavaScript: the used parts. In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. New York, NY, USA: ACM. Pages 109-110. doi:10.1145/2384716.2384762 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate the efficacy of any language design decision.
706. Khilan Gudka, Tim Harris & Susan Eisenbach (2012): Lock Inference in the Presence of Large Libraries. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 308-332. doi:10.1007/978-3-642-31057-7_15 Exclusion reasons: Q1–2 [III.ajk]This article is concerned with an implementation technique, not a language design issue.
707. Tor Guimaraes (1985): A study of application program development techniques. Communications of the ACM 28 (5). Pages 494-499. doi:10.1145/3532.3534 Exclusion reasons: Q1–2 [III.ajk]This article presents a study comparing software development methodologies. There is no PL design issue here.
708. Nuno Guimarães (1991): Building generic user interface tools: an experience with multiple inheritance. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications. Pages 89-96. doi:10.1145/117954.117961 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a lagnuage design decision, at least not comparatively.
709. Sumit Gulwani (2011): Automating string processing in spreadsheets using input-output examples. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 317-330. doi:10.1145/1926385.1926423 Exclusion reasons: Q1–2 [II.ajk]Studies program synthesis, no PL design comparison.
710. R. Gumzej, D. Verber, M. Colnaric, J.P. Babau & J.J. Skubich (2000): An Experiment in Design and Analysis of Real-Time Applications. Journal of Computing and Information Technology 8 (3). Pages 181-195. doi:10.2498/cit.2000.03.02 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
711. Vineet Gupta, Radha Jagadeesan & Prakash Panangaden (1999): Stochastic processes as concurrent constraint programs. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 189-202. doi:10.1145/292540.292558 Exclusion reasons: Q5 [III.ajk]This analytic-constructive study has no aspiration for empiricity.

712. Gopal Gupta, Enrico Pontelli, Khayri A.M. Ali, Mats Carlsson & Manuel V. Hermenegildo (2001): Parallel execution of prolog programs: a survey. ACM Transactions on Programming Languages and Systems 23 (4). Pages 472-602. doi:10.1145/504083.504085 Exclusion reasons: Q1–2 [II.ajk]Discusses implementation issues.

713. Jürg Gutknecht & Eugene Zueff (2003): Zonnon for .NET – A Language and Compiler Experiment. Volume 2789.In Böszörményi, László and Schojer, Peter (ed.) Modular Programming Languages. Lecture Notes in Computer Science. Pages 132-143. doi:10.1007/978-3-540-45213-3_18 Exclusion reasons: Q1–2 [II.ajk]Language exposition, implementation.

714. Jürg Gutknecht, Vladimir Romanov & Eugene Zueff (2005): The Zonnon Project: A .NET Language and Compiler Experiment. Journal of .NET Technologies 3 (1-3). Pages 189-194. http://wscg.zcu.cz/Rotor/Journal/Archive/2005_Vol_3.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

715. John Guttag (1977): Abstract data types and the development of data structures. Communications of the ACM 20 (6). Pages 396-404. doi:10.1145/359605.359618 Exclusion reasons: Q1–2 [II.ajk]Discussion of a software development approach.

716. John V. Guttag, Ellis Horowitz & David R. Musser (1978): Abstract data types and software validation. Communications of the ACM 21 (12). Pages 1048-1064. doi:10.1145/359657.359666 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

717. Mark Guzdial (2011): From science to engineering. Communications of the ACM 54 (2). Pages 37-39. doi:10.1145/1897816.1897831 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

718. Adolfo Guzmán & Harold V. McIntosh (1966): CONVERT. Communications of the ACM 9 (8). Pages 604-615. doi:10.1145/365758.365787 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

719. Kris Gybels & Andy Kellens (2004): An Experiment in Using Inductive Logic Programming to Uncover Pointcuts. Technical report. ftp://soft.vub.ac.be/tech_report/2004/vub-prog-tr-04-10.pdf Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

720. Christian Haack & Erik Poll (2009): Type-Based Object Immutability with Flexible Initialization. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 520-545. doi:10.1007/978-3-642-03013-0_24 Exclusion reasons: Q5 [II.ajk]Type-theoretical study.

721. Florian Haftmann, Cezary Kaliszyk & Walther Neuper (2010): CTP-based programming languages? Considerations about an experimental design. ACM Communications in Computer Algebra 44 (1/2). Pages 27-41. doi:10.1145/1838599.1838621 Exclusion reasons: Q1–2 [II.ajk]Resarch program exposition.

722. Nicholas Haines, Darrell Kindred, J. Gregory Morrisett, Scott M. Nettles & Jeannette M. Wing (1994): Composing first-class transactions. ACM Transactions on Programming Languages and Systems 16 (6). Pages 1719-1736. doi:10.1145/197320.197346 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

723. Elnar Hajiyev, Mathieu Verbaere & Oege de Moor (2006): codeQuest: Scalable Source Code Queries with Datalog. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 2-27. doi:10.1007/11785477_2 Exclusion reasons: Q1–2 [II.ajk]Query language development (no comparison as far as the abstract is concerned).

724. Daniel C. Halbert & Patrick D. O'Brien (1987): Using Types and Inheritance in Object-Oriented Languages. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276 . Pages 20-31. doi:10.1007/3-540-47891-4_3 Exclusion reasons: Q1–2 [II.ajk]Programming methodology study.

725. David Hale (1981): Special purpose languages for behavioural experiments. International Journal of Man-Machine Studies 14 (3). Pages 317-339. doi:10.1016/S0020-7373(81)80061-2 Exclusion reasons: Q5 [II.ajk]No empirical evaluation.

726. Philipp Haller & Martin Odersky (2010): Capabilities for Uniqueness and Borrowing. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 354-378. doi:10.1007/978-3-642-14107-2_17 Exclusion reasons: Q5 [III.ajk]This article includes a brief evaluative section in which (presumably) the authors rewrote some code to use their new construction and then compared the result to the original in terms of how many new annotations had to be made. This is essentially (and concededly) the authors' experience report and is thus excluded.

727. Mark I. Halpern (1968): Programming Languages: Toward a general processor for programming languages. Communications of the ACM 11 (1). Pages 15-25. doi:10.1145/362851.362869 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

728. Robert H. Halstead, Jr. (1985): MULTILISP: a language for concurrent symbolic computation. ACM Transactions on Programming Languages and Systems 7 (4). Pages 501-538. doi:10.1145/4472.4478 Exclusion reasons: Q1–2 [II.ajk]Language exposition

729. Michael Hammer, W. Gerry Howe, Vincent J. Kruskal & Irving Wladawsky (1977): A very high level programming language for data processing applications. Communications of the ACM 20 (10). Pages 832-840. doi:10.1145/359863.359886 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.

730. Matthew A. Hammer, Georg Neis, Yan Chen & Umut A. Acar (2011): Self-adjusting stack machines. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 753-772. doi:10.1145/2048066.2048124 Exclusion reasons: Q1–2 [III.ajk]The main construct under discussion is an intermediate, not programming, language. There is some evaluation of the result in the context of a programming language, but the only measurements provided are about performance. The only way to read this article as evaluating the efficacy of a design decision is to have the efficacy be performance of the resulting program and the design choices being standard C semantics and self-adjustment semantics for C; it's awfully weak.

731. H. Hamza & S. Counsell (2012): Simulation of safety-critical, real-time Java: A case study of dynamic analysis of scoped memory consumption. Simulation Modelling Practice and Theory 25. Pages 172-189. doi:10.1016/j.simpat.2012.02.011 Exclusion reasons: Q1–2 [III.ajk]This article evaluates a variety of ways to use real-time Java; there is no question about language design decisions.

732. S. Hanenberg (2011): A chronological experience report from an initial experiment series on static type systems. In 2nd Workshop on Empirical Evaluation of Software Composition Techniques (ESCOT), Lancaster, UK. http://www.les.inf.puc-rio.br/opus/escot2011/files/07_escot2011.pdf Exclusion reasons: Q1–2 [III.ajk]This article gives a retrospective view on a series of controlled experiments; the results of the experiments are not summarised or consolidated.

733. Per Brinch Hansen (1978): Distributed processes: a concurrent programming concept. Communications of the ACM 21 (11). Pages 934-941. doi:10.1145/359642.359651 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

734. Gary W. Hansen & James V. Hansen (1987): Procedural and non-procedural query languages revisited: a comparison of relational algebra and relational calculus. International Journal of Man-Machine Studies 26 (6). Pages 683-694. doi:10.1016/S0020-7373(87)80060-3 Exclusion reasons: Q1–2 [II.ajk]The languages involved are not programming languages by our definition.

735. Wilfred J. Hansen (1992): Subsequence references: first-class values for substrings. ACM Transactions on Programming Languages and Systems 14 (4). Pages 472-489. doi:10.1145/133233.133234 Exclusion reasons: Q3 [III.ajk]This article includes a brief comparison of Ness with C and Icon, by including an extended example in Ness and then briefly discussing equivalent C and Icon programs. It is debatable whether this qualifies as an empirical study, but it is clear that it isn't documented in sufficient detail.

736. A. Hansen (2011): An Empirical Study of Student Programming Bugs. . Honors thesis. http://digitalcommons.usu.edu/honors/81/ Exclusion reasons: Q1–2 [III.ajk]This honors thesis examines student bugs, and does not involve itself in any language design issues.

737. David R. Hanson (1977): RATSNO—an experiment in software adaptability. Software: Practice and Experience 7 (5). Pages 625-630. doi:10.1002/spe.4380070507 Exclusion reasons: Q1–2 [II.ajk]No comparison.

738. David R. Hanson & Ralph E. Griswold (1978): The SL5 procedure mechanism. Communications of the ACM 21 (5). Pages 392-400. doi:10.1145/359488.359502 Exclusion reasons: Q1–2 [II.ajk]Feature exposition.

739. Stephen José Hanson & Richard R. Rosinski (1985): Programmer perceptions of productivity and programming tools. Communications of the ACM 28 (2). Pages 180-189. doi:10.1145/2786.2791 Exclusion reasons: Q1–2 [III.ajk]This article reports a study comparing tools; there is no programming language design relevance.

740. Eric N. Hanson, Tina M. Harvey & Mark A. Roth (1991): Experiences in DBMS implementation using an object-oriented persistent programming language and a database toolkit. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications. Pages 314-328. doi:10.1145/117954.117978 Exclusion reasons: Q5 [III.ajk]This article reports experience in using the E programming language in implementing database system software. Such articles are explicitly foreclosed by the study protocol.

741. Yasunori Harada, Kenichi Yamazaki & Richard Potter (2001): CCC: User-Defined Object Structure in C. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 118-129. doi:10.1007/3-540-45337-7_7 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

742. Bill C. Hardgrave (1997): Adopting object-oriented technology: Evolution or revolution?. Journal of Systems and Software 37 (1). Pages 19-25. doi:10.1016/S0164-1212(96)00046-5 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

743. David Harel (1980): And/Or Programs: A New Approach to Structured Programming. ACM Transactions on Programming Languages and Systems 2 (1). Pages 1-17. doi:10.1145/357084.357085 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

744. David Harel (1988): On visual formalisms. Communications of the ACM 31 (5). Pages 514-530. doi:10.1145/42411.42414 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

745. David Harel, Assaf Marron, Guy Wiener & Gera Weiss (2011): Behavioral programming, decentralized control, and multiple time scales. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 171-182. doi:10.1145/2095050.2095079 Exclusion reasons: Q5 [III.ajk]No empirical evaluation.

746. Seif Haridi, John-Olof Bauner & Gert Svensson (1981): An implementation and empirical evaluation of the tasking facilities in ADA. SIGPLAN Notices 16 (2). Pages 35-47. doi:10.1145/954269.954274 Exclusion reasons: Q1–2 [III.ajk]This article's evaluative work seems to be targeted to an implementation technique, with no relevance to language design.

747. Seif Haridi, Peter Van Roy, Per Brand, Michael Mehl, Ralf Scheidhauer & Gert Smolka (1999): Efficient logic variables for distributed computing. ACM Transactions on Programming Languages and Systems 21 (3). Pages 569-626. doi:10.1145/319301.319347 Exclusion reasons: Q1–2 [II.ajk]Implementation issue.

748. Mark Harman (2002): Side-Effects Considered Harmful (but Rendered Harmless). In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q3 [III.ajk]Based on the abstract at http://www.ppig.org/workshops/14th-speakers.html, the study is reported in other publications. The most likely candidate is dolado-2003. Since that paper will be considered in due course, this record can be excluded.

749. Mark Harman, David Binkley, Keith Gallagher, Nicolas Gold & Jens Krinke (2009): Dependence clusters in source code. ACM Transactions on Programming Languages and Systems 32 (1). doi:10.1145/1596527.1596528 Exclusion reasons: Q1–2 [III.ajk]This article investigates a program comprehension support method and has little relevance to programming language design.

750. Derin Harmanci, Vincent Gramoli & Pascal Felber (2011): Atomic Boxes: Coordinated Exception Handling with Transactional Memory. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 634-657. doi:10.1007/978-3-642-22655-7_29 Exclusion reasons: Q3 [III.ajk]This article reports, among other things, an evaluative study where several pairs of programs, each pair implementing the same specification using different language designs, were compared by execution speed. As there is no report of how these programs were chosen or written for this study, the reporting is not complete enough for inclusion.

751. Andreas Harnack (1996): Languages and Paradigms - Some Topics to Discuss. In PPIG Student Workshop (1996). (Found in http://ppig.org/workshops/9609s-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not report on a study.

752. Tim Harris & Keir Fraser (2003): Language support for lightweight transactions. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. Pages 388-402. doi:10.1145/949305.949340 Exclusion reasons: Q5 [III.ajk]The empirical evaluation in this article concerns itself only with implementation performance.

753. Tim Harris, Simon Marlow, Simon Peyton-Jones & Maurice Herlihy (2005): Composable memory transactions. In Proceedings of the tenth ACM SIGPLAN symposium on Principles and practice of parallel programming. New York, NY, USA: ACM. PPoPP '05. Pages 48-60. doi:10.1145/1065944.1065952 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

754. Tim Harris (2009): Language constructs for transactional memory. In Proc. 36th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 1. doi:10.1145/1480881.1480883 Exclusion reasons: Q1–2Q5 [II.ajk]Language feature analysis from implementation point of view.

755. Tim Harris, Martin Abadi, Rebecca Isaacs & Ross McIlroy (2011): AC: composable asynchronous IO for native languages. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 903-920. doi:10.1145/2048066.2048134 Exclusion reasons: Q1–2 [III.ajk]The empirical evaluation is purely about performance, offering no insight into efficacy.

756. Warren Harrison & Curtis Cook (1986): Are deeply nested conditionals less readable?. Journal of Systems and Software 6 (4). Pages 335-341. doi:10.1016/0164-1212(86)90003-8 Exclusion reasons: Q1–2 [III.ajk]This article studies a programming style question, not a language design question.

757. W. H. Harrison, P. F. Sweeney & J. J. Shilling (1989): Good news, bad news: experience building software development environment using the object-oriented paradigm. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 85-94. doi:10.1145/74877.74887 Exclusion reasons: Q5 [III.ajk]Experience report.

758. P.H. Hartel & W.G. Vree (1992): Arrays in a lazy functional language – a case study: the fast Fourier transform. In 2nd Arrays, functional languages and parallel systems (ATABLE), Montreal, Canada. pp. 52-66. Publication 841.. Pages 52-66. http://doc.utwente.nl/55744/ Exclusion reasons: Q5 [III.ajk]This article reports a study in which several (systematically differentiated) versions of the FFT algorithm are written, using various capabilities of a lazy functional language in the different versions, and compared as to run-time performance. It can be seen as evaluating the choice of providing arrays in a lazy language However, the paper investigates the analytical implications of that choice, and there is very little nontrivial contingency left in the study design, apart from the optimizations perfomed by the compiler (which are of no relevance for us). Hence, I cannot classify this paper as presenting empirical evidence, as it is defined in this study.

759. Pieter H. Hartel & Willem G. Vree (1994): Experiments with destructive updates in a lazy functional language. Computer Languages 20 (3). Pages 177-192. doi:10.1016/0096-0551(94)90003-5 Exclusion reasons: Q1–2 [III.ajk]This article evaluates implementation techniques, not language design decisions.

760. Thorsten Hartmann, Ralf Jungclaus & Gunter Saake (1992): Aggregation in a behavior oriented object model. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 57-77. doi:10.1007/BFb0053030 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.

761. H. Rex Hartson & Deborah Hix (1989): Toward empirically derived methodologies and tools for human-computer interface development. International Journal of Man-Machine Studies 31 (4). Pages 477-494. doi:10.1016/0020-7373(89)90005-9 Exclusion reasons: Q1–2 [II.ajk]HCI in general, no PL aspect.

762. Niranjan Hasabnis, Ashish Misra & R. Sekar (2012): Light-weight bounds checking. In Proceedings of the Tenth International Symposium on Code Generation and Optimization. New York, NY, USA: ACM. CGO '12. Pages 135-144. doi:10.1145/2259016.2259034 Exclusion reasons: Q1–2 [III.ajk]This article deals with an implementation technique, not a language design decision.

763. Saniya Ben Hassen, Henri E. Bal & Ceriel J. H. Jacobs (1998): A task- and data-parallel programming language based on shared objects. ACM Transactions on Programming Languages and Systems 20 (6). Pages 1131-1170. doi:10.1145/295656.295658 Exclusion reasons: Q1–2 [II.ajk]Language exposition with implementation discussion.

764. A. Hassitt (1967): Data directed input-output in FORTRAN. Communications of the ACM 10 (1). Pages 35-39. doi:10.1145/363018.363056 Exclusion reasons: Q1–2 [III.ajk]This is a language feature exposition.

765. John R. Hauser (1996): Handling floating-point exceptions in numeric programs. ACM Transactions on Programming Languages and Systems 18 (2). Pages 139-174. doi:10.1145/227699.227701 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

766. K. Havelund, M. Ingham & D. Wagner (2010): A Case Study in DSL Development: An Experiment with Python and Scala. In Scala Days 2010. http://www.havelund.com/Publications/scala-days-2010-dsl.pdf Exclusion reasons: Q5 [III.ajk]This article explores the implications of the Scala design, and is thus analytical, not empirical, in nature.

767. M. Haveraaen (2000): Case study on algebraic software methodologies for scientific computing1. Scientific Programming 8. Pages 261–273. http://iospress.metapress.com/content/j2wunlrvnlw4knkm/ Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

768. Wilke Havinga, Lodewijk Bergmans & Mehmet Aksit (2008): Prototyping and Composing Aspect Languages: Using an Aspect Interpreter Framework. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 180-206. doi:10.1007/978-3-540-70592-5_9 Exclusion reasons: Q1–2 [II.ajk]Describes a language design framework.

769. Christopher M. Hayden, Edward K. Smith, Michail Denchev, Michael Hicks & Jeffrey S. Foster (2012): Kitsune: efficient, general-purpose dynamic software updating for C. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 249-264. doi:10.1145/2384616.2384635 Exclusion reasons: Q1–2 [III.ajk]The evaluation in this article are more about proof of concept than efficacy.

770. Frederick Hayes-Roth & John McDermott (1978): An interference matching technique for inducing abstractions. Communications of the ACM 21 (5). Pages 401-411. doi:10.1145/359488.359503 Exclusion reasons: Q1–2 [II.ajk]No PL design relevance.

771. Christopher T. Haynes & Daniel P. Friedman (1987): Embedding continuations in procedural objects. ACM Transactions on Programming Languages and Systems 9 (4). Pages 582-598. doi:10.1145/29873.30392 Exclusion reasons: Q1–2 [III.ajk]This article is an exposition of a language feature.

772. Kaizad B Heerjee & Rubik Sadeghi (1988): Rapid implementation of SQL: a case study using YACC and LEX. Information and Software Technology 30 (4). Pages 228-236. doi:10.1016/0950-5849(88)90083-3 Exclusion reasons: Q1–2 [II.ajk]No comparison.

773. Eric C. R. Hehner (1977): Structuring. In Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 201-205. doi:10.1145/512950.512969 Exclusion reasons: Q5 [III.ajk]This is an analytical work with no aspirations for empiricity.

774. Eric C. R. Hehner (1984): Predicative programming Part I. Communications of the ACM 27 (2). Pages 134-143. doi:10.1145/69610.357988 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

775. Eric C. R. Hehner (1984): Predicative programming Part II. Communications of the ACM 27 (2). Pages 144-151. doi:10.1145/69610.357990 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

776. Phillip Heidegger, Annette Bieniusa & Peter Thiemann (2012): Access permission contracts for scripting languages. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 111-122. doi:10.1145/2103656.2103671 Exclusion reasons: Q1–2 [III.ajk]The evaluation in this article consists of annotating existing code and then seeing how random code changes affect the analysis results. This does not evaluate efficacy.

777. M. Held & W. Mann (2011): An Experimental Analysis of Floating-Point Versus Exact Arithmetic. In 23d Canadian Conference on Computational Geometry 2011. http://2011.cccg.ca/proceedings/ Exclusion reasons: Q1–2 [III.ajk]This article reports a comparison of two different software implementations of real number arithmetic. It does not appear to have programming language design relevance.

778. H. Hellerman (1962): Addressing multidimensional arrays. Communications of the ACM 5 (4). Pages 205-207. doi:10.1145/366920.366943 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

779. H. Hellerman (1964): Experimental personalized array translator system. Communications of the ACM 7 (7). Pages 433-438. doi:10.1145/364520.364573 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

780. Peter Henderson (1972): Derived semantics for some programming language constructs. Communications of the ACM 15 (11). Pages 967-973. doi: 10.1145/355606.361880 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work.

781. P. Henderson & R. Snowdon (1972): An experiment in structured programming. BIT Numerical Mathematics 12 (1). Pages 38-53. doi:10.1007/BF01932672 Exclusion reasons: Q1–2 [II.ajk]Empirical study of a programming practice, no evaluation of language design

782. Robert Henderson & Benjamin Zorn (1994): A comparison of object-oriented programming in four modern languages. Software: Practice and Experience 24 (11). Pages 1077-1095. doi:10.1002/spe.4380241106 Exclusion reasons: Q1–2 [II.ajk]This article is essentially a tutorial, not a language design issue evaluation.

783. J. Henno (1988): User-friendly syntax: design and presentation. International Journal of Man-Machine Studies 28 (5). Pages 551-572. doi: 10.1016/S0020-7373(88)80060-9 Exclusion reasons: Q1–2 [III.ajk]This article presents a controlled experiment with human participants comparing three sublanguages for learnability when they are presented using a metasyntactic notation. This does not seem to have efficacy implications.

784. Roger Henry (1981): Real-time programming languages. International Journal of Man-Machine Studies 14 (3). Pages 355-369. doi:10.1016/S0020-7373(81)80063-6 Exclusion reasons: Q5 [II.ajk]No empirical evaluation.

785. Thomas A. Henzinger (2004): Rich Interfaces for Software Modules. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 516-517. doi:10.1007/978-3-540-24851-4_23 Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.

786. Maurice P. Herlihy & Barbara Liskov (1982): A Value Transmission Method for Abstract Data Types. ACM Transactions on Programming Languages and Systems 4 (4). Pages 527-551. doi:10.1145/69622.357182 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

787. Maurice Herlihy (1993): A methodology for implementing highly concurrent data objects. ACM Transactions on Programming Languages and Systems 15 (5). Pages 745-770. doi:10.1145/161468.161469 Exclusion reasons: Q1–2 [II.ajk]Implementation technique.

788. Maurice Herlihy (2010): Transactional Memory Today. Volume 5966.In Janowski, Tomasz and Mohanty, Hrushikesha (ed.) Distributed Computing and Internet Technology.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 1-12. doi:10.1007/978-3-642-11659-9_1 Exclusion reasons: Q5 Q6 Q7 [II.ajk]This article does not aspire to empiricity.

789. E. H. den Hertog, H J. C. Gerbscheid & M. L. Kersten (1983): DO-SELECT reconsidered. SIGPLAN Notices 18 (3). Pages 32-35. doi:10.1145/988209.988212 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

790. Carl Hewitt, Peter Bishop, Irene Greif, Brian Smith, Todd Matson & Richard Steiger (1973): Actor induction and meta-evaluation. In Proc. 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 153-168. doi:10.1145/512927.512942 Exclusion reasons: Q1–2 [II.ajk]Theoretical development of a programming concept.

791. Anthony J. G. Hey (1990): Experiments in mimd parallelism. Future Generation Computer Systems 6 (3). Pages 185-196. doi:10.1016/0167-739X(90)90018-9 Exclusion reasons: Q1–2 [III.ajk]This selection decision is based on the conference paper doi:10.1007/3-540-51285-3_31, since the journal paper is not readily available to me and it looks like there is no great difference between them. This article does not discuss any PL design issues.

792. Abbas Heydarnoori, Krzysztof Czarnecki & Thiago Tonelli Bartolomei (2009): Supporting Framework Use via Automatically Extracted Concept-Implementation Templates. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 344-368. doi:10.1007/978-3-642-03013-0_16 Exclusion reasons: Q1–2 [II.ajk]No relevance to PL design.

793. T. J. Hickey (1989): CLP and constraint abstraction. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 125-133. doi:10.1145/75277.75288 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

794. Timothy J. Hickey (2000): Analytic constraint solving and interval arithmetic. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 338-351. doi:10.1145/325694.325738 Exclusion reasons: Q1–2 [II.ajk]Language exposition, formal development, a little no-comparison evaluation (based on the abstract).

795. Michael Hicks & Scott Nettles (2005): Dynamic software updating. ACM Transactions on Programming Languages and Systems 27 (6). Pages 1049-1096. doi:10.1145/1108970.1108971 Exclusion reasons: Q1–2 [III.ajk]The new language under discussion (if it can be characterized as such) here is intended for automatic generation and is thus excluded from our definition of PLs.

796. Paul N. Hilfinger (1988): An Ada package for dimensional analysis. ACM Transactions on Programming Languages and Systems 10 (2). Pages 189-203. doi:10.1145/42190.42346 Exclusion reasons: Q1–2 [II.ajk]No comparison; implementation issues.

797. C. J. Hinde (1986): Fuzzy prolog. International Journal of Man-Machine Studies 24 (6). Pages 569-595. doi:10.1016/S0020-7373(86)80009-8 Exclusion reasons: Q5 [II.ajk]Evaluation appears only analytical.

798. Ralf Hinze (2000): A new approach to generic functional programming. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 119-132. doi:10.1145/325694.325709 Exclusion reasons: Q5 [III.ajk]This study explores the implications of existing theoretical assumptions; there is no empirical work reported in it.

799. Tom Hirschowitz & Xavier Leroy (2005): Mixin modules in a call-by-value setting. ACM Transactions on Programming Languages and Systems 27 (5). Pages 857-881. doi:10.1145/1086642.1086644 Exclusion reasons: Q1–2 [II.ajk]Theoretical and implementation study.

800. Martin Hirzel & Robert Grimm (2007): Jeannie: granting java native interface developers their wishes. In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. Pages 19-38. doi:10.1145/1297027.1297030 Exclusion reasons: Q5 [II.ajk]Language exposition.

801. Ellen Hisdal (1981): The IF THEN ELSE statement and interval-valued fuzzy sets of higher type. International Journal of Man-Machine Studies 15 (4). Pages 385-455. doi:10.1016/S0020-7373(81)80051-X Exclusion reasons: Q1–2 [II.ajk]No programming language issues.

802. C. A. R. Hoare (1974): Monitors: an operating system structuring concept. Communications of the ACM 17 (10). Pages 549-557. doi:10.1145/355620.361161 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

803. C. A. R. Hoare (1978): Communicating sequential processes. Communications of the ACM 21 (8). Pages 666-677. doi:10.1145/359576.359585 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

804. Charles Antony Richard Hoare (1981): The emperor's old clothes. Communications of the ACM 24 (2). Pages 75-83. doi:10.1145/358549.358561 Exclusion reasons: Q1–2 [III.ajk]This Turing award lecture does not report a study.

805. J.-M. Hoc (1977): Role of mental representation in learning a programming language. International Journal of Man-Machine Studies 9 (1). Pages 87-105. doi:10.1016/S0020-7373(77)80044-8 Exclusion reasons: Q1–2 [II.ajk]No language design issues.

806. J.-M. Hoc & A. Nguyen-Xuan (1990): Language semantics, mental models and analogy. In Psychology of Programming.Academic. Pages 139-156. Exclusion reasons: Q1–2 [III.ajk]This article does not discuss language design decisions.

807. L. Hochstein, J. Carver, F. Shull, S. Asgari & V. Basili (2005): Parallel Programmer Productivity: A Case Study of Novice Parallel Programmers. In Supercomputing, 2005. Proceedings of the ACM/IEEE SC 2005 Conference. Pages 35. doi:10.1109/SC.2005.53 Exclusion reasons: Q1–2 [II.ajk]Study of programmer behaviour.

808. Kevin J. Hoffman, Harrison Metzger & Patrick Eugster (2011): Ribbons: a partially shared memory programming model. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 289-306. doi:10.1145/2048066.2048091 Exclusion reasons: Q1–2 [III.ajk]The evaluations in this article focus mostly on implementation efficiency. There is a refactoring exercise but it is more of analytical than empirical nature.

809. Christoph M. Hoffmann & Michael J. O'Donnell (1982): Programming with Equations. ACM Transactions on Programming Languages and Sys-

tems 4 (1). Pages 83-112. doi:10.1145/357153.357158 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

810. Martin Hofmann (1992): Formal Development of Functional Programs in Type Theory | A Case Study. ECS-LFCS-92-228 at School of informatics at the University of Edinburgh. http://www.lfcs.inf.ed.ac.uk/reports/92/ECS-LFCS-92-228/index.html Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions, in that there is no comparison.
811. John Hogg & Steven Weiser (1987): OTM: Applying objects to tasks. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 388-393. doi:10.1145/38765.38842 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
812. John Hogg (1991): Islands: aliasing protection in object-oriented languages. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications. Pages 271-285. doi:10.1145/117954.117975 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
813. Uwe Hohenstein (1996): Bridging the gap between C++ and relational databases. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 398-420. doi:10.1007/BFb0053071 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
814. Ian M. Holland (1992): Specifying reusable components using contracts. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 287-308. doi:10.1007/BFb0053043 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
815. David Holmes, James Noble & John Potter (1998): Toward Reusable Synchronisation for Object-Oriented Languages. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 585. doi:10.1007/3-540-49255-0_136 Exclusion reasons: Q5 [III.ajk]This very short article does not aspire to empiricity.
816. Anatol W. Holt (1958): General purpose programming systems. Communications of the ACM 1 (5). Pages 7-9. doi:10.1145/368819.368851 Exclusion reasons: Q1–2 [III.ajk]This remarkable early paper does not present evaluative research.
817. Richard C. Holt, James R. Cordy & David B. Wortman (1982): An Introduction to S/SL: Syntax/Semantic Language. ACM Transactions on Programming Languages and Systems 4 (2). Pages 149-178. doi:10.1145/357162.357164 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
818. Richard C. Holt & James R. Cordy (1988): The Turing programming language. Communications of the ACM 31 (12). Pages 1410-1423. doi:10.1145/53580.53581 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.
819. Yasuaki Honda & Akinori Yonezawa (1988): Debugging Concurrent Systems Based on Object Groups. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 267-282. doi:10.1007/3-540-45910-3_16 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
820. Kohei Honda (1996): Composing processes. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 344-357. doi:10.1145/237721.237802 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
821. Kohei Honda, Nobuko Yoshida & Marco Carbone (2008): Multiparty asynchronous session types. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 273-284. doi:10.1145/1328438.1328472 Exclusion reasons: Q5 [II.ajk]Type-theoretic development.
822. John E. Hopcroft, Joseph K. Kearney & Dean B. Krafft (1991): A Case Study of Flexible Object Manipulation. The International Journal of Robotics Research 10 (1). Pages 41-50. doi:10.1177/027836499101000105 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
823. Aiko Hormann (1961): Computer languages for heuristic processes. Communications of the ACM 4 (3). Pages 138. doi:10.1145/366199.366218 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.
824. J. Horning (1979): A case study in language design: Euclid. Volume 69.In Bauer, Friedrich and Broy, Manfred and Dijkstra, E. and Gerhart, S. and Gries, D. and Griffiths, M. and Guttag, J. and Horning, J. and Owicki, S. and Pair, C. and Partsch, H. and Pepper, P. and Wirsing, M. and Wössner, H. (ed.) Program Construction.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 125-132. doi:10.1007/BFb0014665 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.
825. Haruo Hosoya, Jérôme Vouillon & Benjamin C. Pierce (2005): Regular expression types for XML. ACM Transactions on Programming Languages and Systems 27 (1). Pages 46-90. doi:10.1145/1053468.1053470 Exclusion reasons: Q1–2 [II.ajk]Theory and implementation; and XML is not a PL by our definition.
826. Stephanie Houde & Royston Sellman (1994): In search of design principles for programming environments. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. New York, NY, USA: ACM. CHI '94. Pages 424-430. doi:10.1145/191666.191810 Exclusion reasons: Q1–2 [III.ajk]This article does not deal with any language design decisions.
827. F Houdek, D Ernst & T Schwinn (1999): Comparing structured and object-oriented methods for embedded systems: a controlled experiment. In ICSE'99 Workshop on Empirical Studies of Software Development and Evolution (ESSDE). Pages 75-79. Exclusion reasons: Q3 [III.ajk]No full text can be located; it looks like this article was never actually published (WorldCat does not have a record of any ESSDE 1999 proceedings volume).
828. David Hovemeyer, William Pugh & Jaime Spacco (2002): Atomic Instructions in Java. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 133-154. doi:10.1007/3-540-47993-7_6 Exclusion reasons: Q1–2 [III.ajk]This article presents and analyzes a method for allowing Java programs to use atomic operations. It is doubtful that the proposed method amounts to a PL design decision, as the authors explicitly avoid changing the language, and ihndeed the method consists of a transformation of an implementation's internal representation of the program; however, there is room for dissent, as the method requires a programmer to mark synchronized blocks as eligible for the transformation. In any case, the only possibly empirical evaluation reported involves performance measurements comparing the proposed method to plain Java. Performance is an attribute of implementation and does not (in most cases, including here) bear on the efficacy of the language design decision.
829. Aram Hovsepyan, Riccardo Scandariato, Stefan Van Baelen, Yolande Berbers & Wouter Joosen (2010): From aspect-oriented models to aspect-oriented code?: the maintenance perspective. In Proceedings of the 9th International Conference on Aspect-Oriented Software Development. New York, NY, USA: ACM. AOSD '10. Pages 85-96. doi:10.1145/1739230.1739241 Exclusion reasons: Q1–2 [III.ajk]This article studies process, not language issues.
830. Aram Hovsepyan, Stefan Van Baelen, Riccardo Scandariato, Wouter Joosen & Serge Demeyer (2010): An Experimental Design for Evaluating the Maintainability of Aspect-Oriented Models Enhanced with Domain-Specific Constructs. In Fifteenth international workshop on aspect-oriented modeling (AOM@MoDELS 2010). http://people.cs.kuleuven.be/~aram.hovsepyan/papers/experimental_design.pdf Exclusion reasons: Q1–2 [III.ajk]This article presents a study design, not a completed study.
831. Zhenjiang Hu, Masato Takeichi & Wei-Ngan Chin (1998): Parallelization in calculational forms. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 316-328. doi:10.1145/268946.268972 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
832. Raymond Hu, Nobuko Yoshida & Kohei Honda (2008): Session-Based Distributed Programming in Java. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 516-541. doi:10.1007/978-3-540-70592-5_22 Exclusion reasons: Q5 [III.ajk]There is aspiration for an empirical evaluation of the efficacy of any PL design decision.
833. Shan Shan Huang, David Zook & Yannis Smaragdakis (2007): Morphing: Safely Shaping a Class in the Image of Others. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 399-424. doi:10.1007/978-3-540-73589-2_19 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
834. Shan Shan Huang, Amir Hormati, David F. Bacon & Rodric Rabbah (2008): Liquid Metal: Object-Oriented Programming Across the Hardware/Software Boundary. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 76-103. doi:10.1007/978-3-540-70592-5_5 Exclusion reasons: Q1–2 [II.ajk]No comparison.
835. Wei Huang, Ana Milanova, Werner Dietl & Michael D. Ernst (2012): Reim & ReImInfer: checking and inference of reference immutability and method purity. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 879-896. doi:10.1145/2384616.2384680 Exclusion reasons: Q1–2 [III.ajk]The construct could theoretically be a language design option, but the empirical results are confusing, but do not appear to evaluate the efficacy of any particular design decision.
836. Paul Hudak & Lauren Smith (1986): Para-functional programming: a paradigm for programming multiprocessor systems. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 243-254. doi:10.1145/512644.512667 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
837. M. Elizabeth C. Hull & R. M. McKeag (1984): Communicating Sequential Processes for Centralized and Distributed Operating System Design. ACM Transactions on Programming Languages and Systems 6 (2). Pages 175-191. doi:10.1145/2993.2381 Exclusion reasons: Q1–2 [III.ajk]This paper explores the use of CSP as a design language. There is no PL desing issue here.
838. Joseph Hummel, Laurie J. Hendren & Alexandru Nicolau (1992): Abstract description of pointer data structures: an approach for improving the analysis and optimization of imperative programs. ACM Transactions on Programming Languages and Systems 1 (3). Pages 243-260. doi:10.1145/151640.151644 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
839. Jin H. Hur & Kilnam Chon (1987): Overview of a Parallel Object-Oriented Language CLIX. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 265-273. doi:10.1007/3-540-47891-4_25 Exclusion reasons: Q5 [III.ajk]This article does

not aspire to empiricity.

840. Harry D. Huskey, M. H. Halstead & R. McArthur (1960): NELIAC - dialect of ALGOL. Communications of the ACM 3 (8). Pages 463-468. doi:10.1145/367368.367373 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition with no efficacy issue evident.
841. H. D. Huskey, Ralph Love & Niklaus Wirth (1963): A syntactic description of BC NELIAC. Communications of the ACM 6 (7). Pages 367-375. doi:10.1145/366663.366664 Exclusion reasons: Q1–2 [III.ajk]This is a language specification.
842. Ivan Hybs (1996): Beyond the Interface: A Phenomenological View of Computer Systems Design. Leonardo 29 (3). Pages 215-223. http://www.jstor.org/stable/1576250 Exclusion reasons: Q5 [III.ajk]This article does not evaluate any language design decisions.
843. Urs Hölzle (1993): Integrating Independently-Developed Components in Object-Oriented Languages. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 36-56. doi:10.1007/3-540-47910-4_4 Exclusion reasons: Q5 [III.ajk]This article presents an analysis of problems in reusing object-oriented code. It has no aspiration to empiricity.
844. Einar W. Høst & Bjarte M. Østvold (2009): Debugging Method Names. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 294-317. doi:10.1007/978-3-642-03013-0_14 Exclusion reasons: Q1–2 [II.ajk]Program analysis.
845. Max Ibel, Michael Schmitt, Klaus Schauser & Anurag Acharya (1999): Shared Memory vs Message Passing on SCI: A Case Study Using Split-C. Volume 1734.In Hellwagner, Hermann and Reinefeld, Alexander (ed.) SCI: Scalable Coherent Interface.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 267-280. doi:10.1007/10704208_21 Exclusion reasons: Q1–2 [III.ajk]This article evaluates implementation strategies, not language design decisions.
846. Ali Ibrahim, Yang Jiao, Eli Tilevich & William R. Cook (2009): Remote Batch Invocation for Compositional Object Services. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 595-617. doi:10.1007/978-3-642-03013-0_27 Exclusion reasons: Q5 [III.ajk]The arguably empirical part of this study explicitly evaluates implementation efficiency, not design efficacy.
847. Jean Ichbiah (1984): ADA: past, present, future. Communications of the ACM 27 (10). Pages 990-997. doi:10.1145/358274.358278 Exclusion reasons: Q1–2 [II.ajk]Not a study.
848. Yuuji Ichisugi & Akira Tanaka (2002): Difference-Based Modules: A Class-Independent Module Mechanism. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 62-88. doi:10.1007/3-540-47993-7_3 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
849. Atsushi Igarashi & Benjamin C. Pierce (1999): Foundations for Virtual Types. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 161-185. doi:10.1007/3-540-48743-3_8 Exclusion reasons: Q5 [II.ajk]Type-theoretic investigation.
850. Atsushi Igarashi & Benjamin C. Pierce (2000): On Inner Classes. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 129-153. doi:10.1007/3-540-45102-1_7 Exclusion reasons: Q5 [II.ajk]Formal theoretical development only, based on the abstract.
851. Atsushi Igarashi, Benjamin C. Pierce & Philip Wadler (2001): Featherweight Java: a minimal core calculus for Java and GJ. ACM Transactions on Programming Languages and Systems 23 (3). Pages 306-450. doi:10.1145/503502.503505 Exclusion reasons: Q1–2 [II.ajk]Language exposition, formal theoretical work.
852. Atsushi Igarashi & Mirko Viroli (2002): On Variance-Based Subtyping for Parametric Types. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 449-469. doi:10.1007/3-540-47993-7_19 Exclusion reasons: Q5 [III.ajk]This is a constructive-analytical study with no aspirations to empiricity.
853. Atsushi Igarashi & Mirko Viroli (2006): Variant parametric types: A flexible subtyping scheme for generics. ACM Transactions on Programming Languages and Systems 28 (5). Pages 795-847. doi:10.1145/1152649.1152650 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
854. Juhani Iivari (1995): Object-orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis. Information and Software Technology 37 (3). Pages 155-163. doi:10.1016/0950-5849(95)98926-7 Exclusion reasons: Q1–2 [II.ajk]This article deals with analysis, not language design issues.
855. Satoshi Imai, Takahiro Yamaguchi & Givargis A. Danialy (1992): Which paradigm can improve the reliability of next-generation measurement system software. In Addendum to the proceedings on Object-oriented programming systems, languages, and applications (Addendum). New York, NY, USA: ACM. OOPSLA '92. Pages 153-155. doi:10.1145/157709.157739 Exclusion reasons: Q1–2 [III.ajk]This poster does not appear to evaluate any language design decisions
856. Lintaro Ina & Atsushi Igarashi (2011): Gradual typing for generics. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 609-624. doi:10.1145/2048066.2048114 Exclusion reasons: Q5 [III.ajk]This article offers no empirical evaluation of the efficacy of the construct.
857. Daniel H. H. Ingalls (1978): The Smalltalk-76 programming system design and implementation. In Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 9-16. doi:10.1145/512760.512762 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity.
858. James Ingham & Malcolm Munro (1998): Applying a Domain Specific Language Approach to Component Oriented Programming. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 583. doi:10.1007/3-540-49255-0_36 Exclusion reasons: Q5 [III.ajk]This very short article does not report an empirical study.
859. E. T. Irons & F. S. Acton (1959): A proposed interpretation in ALGOL. Communications of the ACM 2 (12). Pages 14-15. doi:10.1145/368518.368546 Exclusion reasons: Q5 [III.ajk]This brief article analyzes a defect in an early proposal of ALGOL. There is no empiricity involved.
860. Edgar T. Irons (1970): Experience with an extensible language. Communications of the ACM 13 (1). Pages 31-40. doi:10.1145/361953.361966 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
861. Barbara S. Isa, R. James Evey, Bernard W. McVey & Alan S. Neal (1985): An empirical comparison of two metalanguages. International Journal of Man-Machine Studies 23 (3). Pages 215-229. doi:10.1016/S0020-7373(85)80033-X Exclusion reasons: Q1–2 [II.ajk]Compares syntactic metalanguages.
862. Yutaka Ishikawa & Mario Tokoro (1986): A concurrent object-oriented knowledge representation language Orient84/K: its features and implementation. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 232-241. doi:10.1145/28697.28720 Exclusion reasons: Q1–2 [III.ajk]This article provides a brief language exposition and then constructs and evaluates its implementation. The language itself is not subject to evaluation.
863. Yutaka Ishikawa & Hideyuki Tokuda (1990): Object-oriented real-time language design: constructs for timing constraints. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 289-298. doi:10.1145/97945.97980 Exclusion reasons: Q1–2 [III.ajk]This paper is a language exposition, with very little attempt at comparative evaluation.
864. Yutaka Ishikawa (1992): Communication mechanism on autonomous objects. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications. Pages 303-314. doi:10.1145/141936.141962 Exclusion reasons: Q5 [III.ajk]This article does no aspire to empiricity.
865. Kenneth E. Iverson (1964): Formalism in programming languages. Communications of the ACM 7 (2). Pages 80-88. doi:10.1145/363921.363933 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
866. Kenneth E. Iverson (1979): Operators. ACM Transactions on Programming Languages and Systems 1 (2). Pages 161-176. doi:10.1145/357073.357074 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
867. Kenneth E. Iverson (1980): Notation as a tool of thought. Communications of the ACM 23 (8). Pages 444-465. doi:10.1145/358896.358899 Exclusion reasons: Q5 [III.ajk]This article is mathematically oriented and does not present any empirical work.
868. Jonathan P. Jacky & Ira J. Kalet (1987): An object-oriented programming discipline for standard Pascal. Communications of the ACM 30 (9). Pages 772-776. doi:10.1145/30401.30403 Exclusion reasons: Q1–2 [III.ajk]This article introduces a style of programming in Pascal and thus presents no PL design issue.
869. Bart Jacobs, Frank Piessens, Jan Smans, K. Rustan M. Leino & Wolfram Schulte (2008): A programming model for concurrent object-oriented programs. ACM Transactions on Programming Languages and Systems 31 (1). doi:10.1145/1452044.1452045 Exclusion reasons: Q1–2 [II.ajk]Studies program analysis.
870. Bart Jacobs & Frank Piessens (2009): Failboxes: Provably Safe Exception Handling. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 470-494. doi:10.1007/978-3-642-03013-0_22 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
871. M. Jadud, B. Chenoweth & J. Scheleter (2003): Little Languages for Little Robots. In PPIG 2003. (Found in http://ppig.org/workshops/15th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.
872. J. Jaffar & J.-L. Lassez (1987): Constraint logic programming. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 111-119. doi:10.1145/41625.41635 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

873. Joxan Jaffar, Spiro Michaylov, Peter J. Stuckey & Roland H. C. Yap (1992): The CLP(R) language and system. ACM Transactions on Programming Languages and Systems 14 (3). Pages 339-395. doi:10.1145/129393.129398 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

874. Suresh Jagannathan & Gul Agha (1992): A reflective model of inheritance. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 350-371. doi:10.1007/BFb0053046 Exclusion reasons: Q5 [III.ajk]This article presents and analyzes a way to model inheritance. It has no aspiration to empiricity.

875. Suresh Jagannathan (1994): Metalevel building blocks for modular systems. ACM Transactions on Programming Languages and Systems 16 (3). Pages 456-492. doi:10.1145/177492.177578 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

876. Paul J. Jalics (1984): Cobol vs. PL/1: some performance comparisons. Communications of the ACM 27 (3). Pages 216-221. doi:10.1145/357994.358019 Exclusion reasons: Q1–2 [III.ajk]This article presents a simple comparison of performance of a Cobol and a PL/1 implementation on a specific machine. As such, it does not present a PL design issue.

877. Paul J. Jalics (1987): COBOL on a PC: a new perspective on a language and its performance. Communications of the ACM 30 (2). Pages 142-154. doi:10.1145/12527.12530 Exclusion reasons: Q1–2 [II.ajk]Studies implementations.

878. Patrik Jansson & Johan Jeuring (1997): PolyP—a polytypic programming language extension. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 470-482. doi:10.1145/263699.263763 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

879. Doug Janzen & Kris De Volder (2004): Programming with Crosscutting Effective Views. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 197-220. doi:10.1007/978-3-540-24851-4_9 Exclusion reasons: Q5 [III.ajk]This article introduces and discusses a new construct with no aspiration for epiricity.

880. D. Jarecka, S. Arabas, M. Fijalkowski & A. Gaynor (2012): On the tradeoffs of programming language choice for numerical modelling in geoscience. A case study comparing modern Fortran, C++/Blitz++ and Python/NumPy.. Volume 14.In Abbasi, A. and Giesen, N. (ed.) EGU General Assembly Conference Abstracts. EGU General Assembly Conference Abstracts. Pages 680. http://adsabs.harvard.edu/abs/2012EGUGA..14..680J Exclusion reasons: Q5 [II.ajk]Implementation efficiency compared using one algorithm, very likely not empirical in our sense.

881. Ciera Jaspan & Jonathan Aldrich (2009): Checking Framework Interactions with Relationships. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 27-51. doi:10.1007/978-3-642-03013-0_3 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work.

882. Stan Jefferson & Samuel N. Kamin (1986): Executable specifications with quantifiers in the FASE system. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 318-326. doi:10.1145/512644.512674 Exclusion reasons: Q1–2 [II.ajk]Even assuming there's a PL there, there's no comparison.

883. C. Wesley Jenkins (1981): Application prototyping: A case study. In Proceedings of the 1981 ACM workshop/symposium on Measurement and evaluation of software quality. New York, NY, USA: ACM. Pages 21-27. doi:10.1145/800003.807905 Exclusion reasons: Q1–2 [III.ajk]This article has no relevance to programming language design.

884. T. Jensen, D. Le Métayer & T. Thorn (1998): Coarse Grained Java Security Policies. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 584. doi:10.1007/3-540-49255-0_75 Exclusion reasons: Q5 [III.ajk]This article does not appear to describe an empirical study.

885. Chang-Hyan Jo (1996): An experiment on a concurrent object-oriented programming language. In Proceedings of the 1996 ACM symposium on Applied Computing. Pages 98–104. doi:10.1145/331119.331157 Exclusion reasons: Q5 [III.ajk]This article, despite its title, does not aspire to empiricity.

886. G. F. Johnson & D. Duggan (1988): Stores and partial continuations as first-class objects in a language and its environment. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 158-168. doi:10.1145/73560.73574 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

887. Richard Johnson & Murugappan Palaniappan (1993): MetaFlex: A Flexible Metaclass Generator. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 502-527. doi:10.1007/3-540-47910-4_25 Exclusion reasons: Q5 [III.ajk]This article, though it includes an example styled as a "case study", does not aspire to empiricity.

888. Richard A. Johnson, Bill C. Hardgrave & E. Reed Doke (1999): An industry analysis of developer beliefs about object-oriented systems development. SIGMIS Database 30 (1). Pages 47-64. doi:10.1145/342251.342263 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

889. John B. Johnston (1965): A class of unambiguous computer languages. Communications of the ACM 8 (3). Pages 147-149. doi:10.1145/363791.363796 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

890. Dan Johnston & Andrew Lister (1980): An experiment in software science. Volume 79.In Language Design and Programming Methodology. Pages 195-215. doi:10.1007/3-540-09745-7_15 Exclusion reasons: Q1–2 [III.ajk]This article presents a study evaluating a theory of program metrics and does not evaluate a language design decision.

891. Anita K. Jones & Barbara H. Liskov (1978): A language extension for expressing constraints on data access. Communications of the ACM 21 (5). Pages 358-367. doi:10.1145/359488.359493 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

892. Michael B. Jones & Richard F. Rashid (1986): Mach and Matchmaker: kernel and language support for object-oriented distributed systems. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 67-77. doi:10.1145/28697.28705 Exclusion reasons: Q1–2 [III.ajk]This system exposition article does not evaluate language design decisions.

893. Mark P. Jones (1996): Using parameterized signatures to express modular structure. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 68-78. doi:10.1145/237721.237731 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

894. Mark P. Jones (1997): First-class polymorphism with type inference. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 483-496. doi:10.1145/263699.263765 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

895. S. Joosten (1987): Modelling and Simulation in a Functional Programming Language: A Case Study. University of Twente, Department of Informatics. (No URL known.) Exclusion reasons: Q1–2Q5 [III.ajk]There is no comparison language, and the evaluation is analytic in nature.

896. David Jordan (1990): Implementation benefits of C++ language mechanisms. Communications of the ACM 33 (9). Pages 61-64. doi:10.1145/83880.84460 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

897. Philippe Jorrand (1986): Term rewriting as a basis for the design of a functional and parallel programming language. Volume 232.In Bibel, Wolfgang and Jorrand, Philippe (ed.) Fundamentals of Artificial Intelligence.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 220-276. doi:10.1007/BFb0022684 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

898. Yang Jun, Greg Michaelson & Phil Trinder (2000): How do people check polymorphic types?. In PPIG 2000. (Found in http://ppig.org/workshops/12th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article reports a study about the behaviour of human experts when they infer types to polymorphic functions in a HM type system. As such it does not evaluate a language design decision.

899. Esa-Matti Järvinen (1998): The Lego/Logo Learning Environment in Technology Education: An Experiment in a Finnish Context. Journal of Technology Education 9 (2). http://scholar.lib.vt.edu/ejournals/JTE/v9n2/jrvinen.html Exclusion reasons: Q1–2 [III.ajk]The study reported here does not evaluate language design decisions.

900. J. -M. Jézéquel (1992): EPEE: an eiffel environment to program distributed memory parallel computers. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 197-212. doi:10.1007/BFb0053038 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision. (Note that the article itself says so – on page 199, in the last paragraph before Section 2: "... to embed data parallelism in [an object-oriented language] in a clean and elegant fashion, without modifying the [language's] syntax and semantic [sic!].")

901. J. -M. Jézéquel (1993): Transparent parallelisation through reuse: between a compiler and a library approach. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 384-405. doi:10.1007/3-540-47910-4_20 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

902. Jean-Marc Jézéquel & Jean-Lin Pacherie (1996): Parallel operators. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 275-294. doi:10.1007/BFb0053066 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

903. Dennis G. Kafura & Keung Hae Lee (1989): Inheritance in Actor Based Concurrent Object-Oriented Languages. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 131-145. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890131.pdf Exclusion reasons: Q5 [III.ajk]This article has no empirical aspirations.

904. Dona M. Kagan & John M. Douthat (1985): Personality and learning FORTRAN. International Journal of Man-Machine Studies 22 (4). Pages 395-402. doi:10.1016/S0020-7373(85)80046-8 Exclusion reasons: Q1–2 [II.ajk]This article evaluates the relationship between personality and learning programming.

905. Kenneth Kahn, Eric Dean Tribble, Mark S. Miller & Daniel G. Bobrow (1986): Objects in concurrent logic programming languages. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 242-257. doi:10.1145/28697.28721 Exclusion

reasons: Q5 [III.ajk]This article does not aspire to empiricity.

906. R. Y. Kain (1969): Block structures, indirect addressing, and garbage collection. Communications of the ACM 12 (7). Pages 395-398. doi:10.1145/363156.363175 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

907. Gail E. Kaiser & Brent Hailpern (1992): An object-based programming model for shared data. ACM Transactions on Programming Languages and Systems 14 (2). Pages 201-264. doi:10.1145/128861.128866 Exclusion reasons: Q1–2 [II.ajk]Language feature exposition.

908. Laxmikant V. Kale & Sanjeev Krishnan (1993): CHARM++: a portable concurrent object oriented system based on C++. In OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. Pages 91-108. doi:10.1145/165854.165874 Exclusion reasons: Q5 [III.ajk]This article does not evaluate empirically any language design decisions.

909. LA Kalinichenko (1968): SLANG–Computer description and simulation-oriented experimental programming language. Simulation Programming Languages, North-Holland, Amsterdam. Pages 101–115. (No URL known.) Exclusion reasons: Q1–2 [II.ajk]Based on the title only, a language exposition.

910. L. Kalé, M. Bhandarkar, R. Brunner, N. Krawetz, J. Phillips & A. Shinozaki (1998): NAMD: A case study in multilingual parallel programming. Volume 1366.In Li, Zhiyuan and Yew, Pen-Chung and Chatterjee, Siddharta and Huang, Chua-Huang and Sadayappan, P. and Sehr, David (ed.) Languages and Compilers for Parallel Computing.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 367-381. doi:10.1007/BFb0032705 Exclusion reasons: Q1–2 [III.ajk]This article describes a programming project using several languages in the same project. There is no attempt at evaluating the language designs.

911. Y. Kambayashi & H. F. Ledgard (2005): An experiment on a new programming paradigm. In Computational Cybernetics, 2005. ICCC 2005. IEEE 3rd International Conference on. Pages 155-160. doi:10.1109/ICCCYB.2005.1511566 Exclusion reasons: Q1–2 [III.ajk]This article reports a controlled experiment evaluating two different styles of programming. Since both styles used the same programming language, there was no issue of language design.

912. Samuel Kamin (1980): Final data type specifications: a new data type specification method. In Proc. 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 131-138. doi:10.1145/567446.567459 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

913. Nozomu Kaneko & Takehisa Onisawa (2005): An Experimental Study on Computer Programming with Linguistic Expressions. Volume 3681.In Khosla, Rajiv and Howlett, Robert and Jain, Lakhmi (ed.) Knowledge-Based Intelligent Information and Engineering Systems.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 154-154. doi:10.1007/11552413_130 Exclusion reasons: Q1–2 [III.ajk]This article evaluates a programming language but there is no comparison.

914. H. Kanner (1959): An algebraic translator. Communications of the ACM 2 (10). Pages 19-22. doi:10.1145/368453.368461 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

915. M. Kantamneni (2011): JINI–Literature Review. Student project report. http://salsahpc.indiana.edu/b534projects/sites/default/files/public/3_JINI_Kantamneni,%20Manish.pdf Exclusion reasons: Q1–2 [II.ajk]No language design issue.

916. Michael Karr & III David B. Loveman (1978): Incorporation of units into programming languages. Communications of the ACM 21 (5). Pages 385-391. doi:10.1145/359488.359501 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.

917. Lora L. Kassab & Jeffrey Voas (1998): Agent Trustworthiness. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 588. doi:10.1007/3-540-49255-0_78 Exclusion reasons: Q5 [III.ajk]This is a one-page abstract of a paper (not published here) that does not appear to have empirical content.

918. Roslaili Kassim, Nordin Abu Bakar & Khalil Hj Awang (2008): Application performance benchmark: An experimental analysis on C# programs. Volume 1.In Information Technology, 2008. ITSim 2008. International Symposium on. Pages 1-5. doi:10.1109/ITSIM.2008.4631616 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

919. L. C. L. Kats & E. Visser (2010): Encapsulating Software Platform Logic by Aspect-Oriented Programming: A Case Study in Using Aspects for Language Portability. In Source Code Analysis and Manipulation (SCAM), 2010 10th IEEE Working Conference on. Pages 147-156. doi:10.1109/SCAM.2010.11 Exclusion reasons: Q1–2 [II.ajk]No PL DD evaluation.

920. Jesse H. Katz & William C. McGee (1963): An experiment in non-procedural programming. In Proceedings of the November 12-14, 1963, fall joint computer conference. Pages 1-13. doi:10.1145/1463822.1463824 Exclusion reasons: Q5 [III.ajk]This early paper explores the possibility of writing a program in a declarative language. As such, it explores the implications of a given fact, and is not empirical.

921. Shmuel Katz (1993): A superimposition control construct for distributed systems. ACM Transactions on Programming Languages and Systems 15 (2). Pages 337-356. doi:10.1145/169701.169682 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

922. Jacob Katzenelson (1983): Higher level programming and data abstractions—a case study using enhanced C. Software: Practice and Experience 13 (7). Pages 577–595. doi:10.1002/spe.4380130703 Exclusion reasons: Q1–2 [II.ajk]No comparison

923. William H. Kautz, Edward A. Voorhees & T. A. Jeeves (1958): Automatic programming systems. Communications of the ACM 1 (8). Pages 6-8. doi:10.1145/368892.368910 Exclusion reasons: Q1–2 [III.ajk]This publication is not at all relevant to our study.

924. Chuanle Ke, Lei Liu, Chao Zhang, Tongxin Bai, Bryan Jacobs & Chen Ding (2011): Safe parallel programming using dynamic dependence hints. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 243-258. doi:10.1145/2048066.2048087 Exclusion reasons: Q1–2 [III.ajk]The evaluation here is just about implementation performance.

925. Aaron William Keen (2002): Integrating concurrency constructs with object-oriented programming languages: a case study. . PhD at University of California, Davis. http://www.cs.ucdavis.edu/~olsson/research/jr/papers/keen.ps Exclusion reasons: Q5 Disagreement resolution result. [sel-2.kaijanaho]No empiricity beyond a brief performance study using benchmarks.

926. Aaron Keen & Ronald Olsson (2002): Exception Handling during Asynchronous Method Invocation. Volume 2400.In Monien, Burkhard and Feldmann, Rainer (ed.) Euro-Par 2002 Parallel Processing.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 337-412. doi:10.1007/3-540-45706-2_90 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

927. Aaron W. Keen, Tingjian Ge, Justin T. Maris & Ronald A. Olsson (2004): JR: Flexible distributed programming in an extended Java. ACM Transactions on Programming Languages and Systems 26 (3). Pages 578-608. doi:10.1145/982158.982162 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

928. Caitlin Kelleher & Randy Pausch (2005): Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. ACM Computing Surveys 37 (2). Pages 83-137. doi:10.1145/1089733.1089734 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

929. Andy Kellens, Kim Mens, Johan Brichau & Kris Gybels (2006): Managing the Evolution of Aspect-Oriented Software with Model-Based Pointcuts. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 501-525. doi:10.1007/11785477_28 Exclusion reasons: Q5 [III.ajk]The evaluation in this article is analytical in nature.

930. Roy F. Keller (1977): On control constructs for constructing programs. SIGPLAN Notices 12 (9). Pages 36-44. doi:10.1145/954604.954606 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

931. Robert M. Keller & M. R. Sleep (1986): Applicative caching. ACM Transactions on Programming Languages and Systems 8 (1). Pages 88-108. doi:10.1145/5001.5004 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

932. R. A. Kelley (1973): APLGOL, an Experimental Structured Programming Language. IBM Journal of Research and Development 17 (1). Pages 69-73. doi:10.1147/rd.171.0069 Exclusion reasons: Q5 [II.ajk]Language exposition; based on the abstract, the evaluation is "outlined", and there does not appear to be any actual data collection to support that.

933. James Kempf, Warren Harris, Roy D'Souza & Alan Snyder (1987): Experience with CommonLoops. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 214-226. doi:10.1145/38765.38827 Exclusion reasons: Q3 [III.ajk]This article presents two empirical studies attempting to evaluate the efficacy of CommonLoops, with Common Lisp as the control. The studies are reported mostly through statements of results, and the methodology of the studies is left mostly up to the reader's imagination. Thus, this article cannot be considered a "complete" report. No other report of the study has been identified.

934. Elizabeth A. Kendall (1999): Role model designs and implementations with aspect-oriented programming. In OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 353-369. doi:10.1145/320384.320423 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

935. Andrew J. Kennedy (1997): Relational parametricity and units of measure. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 442-455. doi:10.1145/263699.263761 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

936. Y. Kermarrec, L. Nana & L. Pautet (1994): Implementing an efficient fault tolerance mechanism in Ada 9X: an early experiment with GNAT. In Ada Belgium conference in Brussells. ("printed in the Ada-Belgium Newsletter Volume 3, Numbers 2+3, Winter 1994, November 1994.") http://public.enst-bretagne.fr/~kermarre/publi/bruxells.ps.gz Exclusion reasons: Q1–2 [III.ajk]This tool exposition paper does not discuss PL design issues.

937. R. K. Kerr & D. B. Percival (1987): Use of object-oriented programming in a time series analysis system. In Conference Proceedings on Object-Oriented

Programming Systems, Languages and Applications (OOPSLA87). Pages 1-10. doi:10.1145/38765.38808 Exclusion reasons: Q1–2 [III.ajk]This article is a reflective project description, presenting no empirical evaluation of programming language design issues.

938. Mik Kersten & Gail C. Murphy (1999): Atlas: a case study in building a web-based learning environment using aspect-oriented programming. In OOPSLA '99: Proceedings of the 14th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 340-352. doi:10.1145/320384.320421 Exclusion reasons: Q1–2 [III.ajk]This article describes the use of AOP in the construction of an actual software system. There was no comparison language, and hence there was no PL design issue present.
939. J. L. W. Kessels (1977): A conceptual framework for a nonprocedural programming language. Communications of the ACM 20 (12). Pages 906-913. doi:10.1145/359897.359900 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
940. A. J. Kfoury, J. Tiuryn & P. Urzyczyn (1988): A proper extension of ML with an effective type-assignment. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 58-69. doi:10.1145/73560.73565 Exclusion reasons: Q5 [II.ajk]Type-theoretical study.
941. Babak Khazaei, Jawed Siddiqi, Andreas Harnack, Rick Osborn & Chris Roast (1996): Further Investigations into the transfer effect of moving from procedural to logic programming. In PPIG 1996. (Found in http://ppig.org/workshops/8th-programme.html.) Exclusion reasons: Q3 [III.ajk]Interlibrary loan service was unable to locate a copy. See msgid 04E17A7BB138C642A2B06A667D96AA1E256EABE2@mbs1.ad.jyu.fi in misc/interlibrary-service-rejects.mbox
942. Babak Khazaei & Chris Roast (2001): The Usability of Formal Specification Representations. In PPIG 2001. (Found in http://ppig.org/workshops/13th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article studies Z, which is not a programming language by our definition.
943. B. Khazaei & M. Jackson (2002): Is there any difference in novice comprehension of a small program written in the event-driven and object-oriented styles?. In Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on. Pages 19-26. doi:10.1109/HCC.2002.1046336 Exclusion reasons: Q1–2 [III.ajk]This article presents a study that attempts to describe the qualitative difference in comprehension between two programming styles. It does not evaluate any language design decisions.
944. Gregor Kiczales, John Lamping, Anurag Mendhekar, Chris Maeda, Cristina Lopes, Jean-Marc Loingtier & John Irwin (1997): Aspect-oriented programming. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 220-242. doi:10.1007/BFb0053381 Exclusion reasons: Q5 Disagreement resolution result. [III.ajk]This article does not aspire to empiricity. [sel-2.kaijanaho]No empiricity. [sel-2.tirronen]Presents arguments against some features, but the goal is to introduce a new language design
945. Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm & William G. Griswold (2001): An Overview of AspectJ. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 327-354. doi:10.1007/3-540-45337-7_18 Exclusion reasons: Q5 [III.ajk]This article introduces AspectJ, a new AOP language. It includes a section that purports to evaluate its efficacy, but the evaluation is analytical or anecdotal.
946. Gregor Kiczales & Mira Mezini (2005): Separation of Concerns with Procedures, Annotations, Advice and Pointcuts. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 195-213. doi:10.1007/11531142_9 Exclusion reasons: Q1–2 [II.ajk]Formal development of understanding and usage guidelines, no evaluation of efficacy
947. Richard B. Kieburtz & Abraham Silberschatz (1979): Comments on "Communicating Sequential Processes". ACM Transactions on Programming Languages and Systems 1 (2). Pages 218-225. doi:10.1145/357073.357077 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
948. Richard B. Kieburtz & Abraham Silberschatz (1983): Access-Right Expressions. ACM Transactions on Programming Languages and Systems 5 (1). Pages 78-96. doi:10.1145/357195.357201 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
949. Jörg Kienzle & Rachid Guerraoui (2002): AOP: Does It Make Sense? The Case of Concurrency and Failures. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 37-61. doi:10.1007/3-540-47993-7_2 Exclusion reasons: Q5 [III.ajk]This article, although it uses the word "experiment" to characterize itself, describes an analytical study of wether AOP can be used to transform a non-transactional program into a transactional one. There is no empiricality involved.
950. Gary A. Kildall & Alan B. Roberts (1972): ALGOL-E: An Experimental Approach to The Study of Programming Languages. In Proceedings of the second SIGCSE technical symposium on Education in computer science. New York, NY, USA: ACM. Pages 127-135. doi:10.1145/800155.805016 Exclusion reasons: Q1–2 [II.ajk]Teaching, language exposition
951. Miryung Kim, Vibha Sazawal, David Notkin & Gail Murphy (2005): An empirical study of code clone genealogies. In Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering. New York, NY, USA: ACM. ESEC/FSE-13. Pages 187-196. doi:10.1145/1081706.1081737 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
952. Aaron Kimball & Dan Grossman (2007): Software Transactions Meet First-Class Continuations. In Proc. Workshop on Scheme and Functional Programming. http://www.schemeworkshop.org/2007/procPaper5.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity, except so far as to evaluating implementation efficiency.
953. David J. King & John Launchbury (1995): Structuring depth-first search algorithms in Haskell. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 344-354. doi:10.1145/199448.199530 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
954. James D. Kiper, Brent Auernheimer & Charles K. Ames (1997): Visual Depiction of Decision Statements: What is Best for Programmers and Non-Programmers?. Empirical Software Engineering 2 (4). Pages 361-379. doi:10.1023/A:1009797801907 Exclusion reasons: Q1–2 [II.ajk]Compares textual to graphical language; since graphical language is not a PL in our definition, there effectively is no comparison for our purposes.
955. Yuliyan Kiryakov & John Galletly (2003): Aspect-oriented programming: case study experiences. In Proceedings of the 4th international conference conference on Computer systems and technologies: e-Learning. New York, NY, USA: ACM. Pages 184-189. doi:10.1145/973620.973651 Exclusion reasons: Q3 Q5 [III.ajk]This article reports a non-Yinite case study in which a program was written using AspectJ, in order to demonstrate the advantages of AOP. The study, charitably described, brings to light consequences of the concept of aspect-oriented programming, and arguably does not concern itself with contingent aspects of the world; thus, it arguably is not an empirical study. Nevertheless, even if one were to consider it empirical, the report itself does not describe the study in sufficient detail to allow the evaluation of its quality. A more complete report has not been identified.
956. Marja-Riita Kivi & Tapio Grönfors (1998): Empirical studies of programmers using continuous display capturing . In PPIG 1998. (Found in http://ppig.org/workshops/10th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not report a study, it introduces a tool. The related work does not summarise or consolidate research that is relevant to us (it focuses more on their methodological approach).
957. Nils Klarlund & Michael I. Schwartzbach (1993): Graph types. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 196-205. doi:10.1145/158511.158628 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
958. Alon Kleinman, Yael Moscowitz, Amir Pnueli & Ehud Sharpio (1991): Communication with directed logic variables. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 221-232. doi:10.1145/99583.99615 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
959. Melvin Klerer (1991): Design of very high-level computer languages: a user oriented approach (2nd ed.). New York, NY, USA: McGraw-Hill, Inc.. Exclusion reasons: Q6 [III.ajk]This book does not aspire to empiricity.
960. Robert J Klerer, Melvin Klerer & Fred Grossman (1992): A language for automated programming of mathematical applications. Computer Languages 17 (3). Pages 169-184. doi:10.1016/0096-0551(92)90027-K Exclusion reasons: Q1–2 [II.ajk]This article does not aspire to empiricity.
961. Carol Diane Klingler (1993): A case study in process definition. In Proceedings of the conference on TRI-Ada '93. New York, NY, USA: ACM. Pages 65-79. doi:10.1145/170657.170682 Exclusion reasons: Q1–2 [III.ajk]This article reports a case study in the use of process definition languages. They are not programming languages as they are primarily meant to be human-read and only incidentally automatically executed.
962. Paul Klint (1980): An overview of the SUMMER programming language. In Proc. 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 47-55. doi:10.1145/567446.567451 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
963. Markus Knasmüller (1998): Oberon-D = Object-Oriented System + Object-Oriented Database. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 587. doi:10.1007/3-540-49255-0_187 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.
964. Claude Y. Knaus (2008): Essential programming paradigm. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 823-826. doi:10.1145/1449814.1449873 Exclusion reasons: Q1–2 [III.ajk]This article does not report a study.
965. Günter Kniesel (1999): Type-Safe Delegation for Run-Time Component Adaptation. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 351-366. doi:10.1007/3-540-48743-3_16 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
966. Kenneth C. Knowlton (1966): A programmer's description of L6. Communications of the ACM 9 (8). Pages 616-625. doi:10.1145/365758.365792 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

967. Donald E. Knuth (1959): RUNCIBLE - algebraic translation on a limited computer. Communications of the ACM 2 (11). Pages 18-21. doi:10.1145/368481.368507 Exclusion reasons: Q5 [III.ajk]This compiler exposition does not aspire to empiricity.
968. Donald E. Knuth & Jack N. Merner (1961): ALGOL 60 confidential. Communications of the ACM 4 (6). Pages 268-272. doi:10.1145/366573.366599 Exclusion reasons: Q5 [II.ajk]From the abstract: "Remarks are based on the authors' interpretations of the ALGOL 60 Report."
969. D. E. Knuth (1964): A proposal for input-output conventions in ALGOL 60. Communications of the ACM 7 (5). Pages 273-283. doi:10.1145/364099.364222 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
970. Donald E. Knuth (1967): The remaining trouble spots in ALGOL 60. Communications of the ACM 10 (10). Pages 611-618. doi:10.1145/363717.363743 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
971. Donald E. Knuth (1971): An empirical study of FORTRAN programs. Software: Practice and Experience 1 (2). Pages 105-133. doi:10.1002/spe.4380010203 Exclusion reasons: Q1–2 [II.ajk]Studies language usage patterns; no direct PL design issue.
972. Roman Knöll & Mira Mezini (2006): Pegasus: first steps toward a naturalistic programming language. In OOPSLA '06: Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications. Pages 542-559. doi:10.1145/1176617.1176628 Exclusion reasons: Q1–2 [III.ajk]This article does not aspire to primary empiricity; and while it does summarise existing empirical research, it focuses on exactly one prior empirical study relevant to us and thus does not qualify as a literature review in any real sense.
973. Roman Knöll & Mira Mezini (2009): π: a pattern language. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications. Pages 503-522. doi:10.1145/1640089.1640128 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
974. Naoki Kobayashi (1999): Quasi-linear types. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 29-42. doi:10.1145/292540.292546 Exclusion reasons: Q1–2 [III.ajk]This article studies an implementation technique (albeit one based on techniques also usable in language design) and thus is outside our scope.
975. Naoki Kobayashi & Davide Sangiorgi (2010): A hybrid type system for lock-freedom of mobile processes. ACM Transactions on Programming Languages and Systems 32 (5). doi:10.1145/1745312.1745313 Exclusion reasons: Q1–2 [II.ajk]Formal theory development.
976. Ján Kollár & Marcel Tóth (2005): An Experiment with Aspect Programming Language. In Proceedings of the 3rd Slovakian-Hungarian Joint Symposium on Applied Machine Intelligence, Herl'any, Slovakia, January. Pages 21–22. http://www.bmf.hu/conferences/SAMI2005/Kollar.pdf Exclusion reasons: Q3 [III.ajk]Selection decision was made based on the full-text copy at http://web.archive.org/web/20081204053959/http://bmf.hu/conferences/SAMI2005/Kollar.pdf (the original URL not being accessible at this time). This article discusses the use of aspects in a newish programming language, and gives a very brief note of an "experiment" (which isn't actually an experiment) in defining and implementing a similar language. The article is generally analytically oriented, with very limited empirical content. For the empirical part, the article is far from a complete report, and I have not been able to find a complete one.
977. Joseph A. Konstan & Lawrence A. Rowe (1991): Developing a GUIDE using object-oriented programming. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications. Pages 75-88. doi:10.1145/117954.117960 Exclusion reasons: Q1–2 [III.ajk]This article describes a study in which a particular application was implemented in a particular language, with conclusions being drawn as to that language's efficacy. Without an actual comparison, it does not evaluate any language design decisions.
978. Timothy D. Korson & Vijay K. Vaishnavi (1986): An empirical study of the effects of modularity on program modifiability. In Papers presented at the first workshop on empirical studies of programmers on Empirical studies of programmers. Pages 168-186. Exclusion reasons: Q1–2 [II.ajk]This article studies programming style, not language design issues.
979. Tomaž Kosar, Marjan Mernik & Jeffrey Carver (2012): Program comprehension of domain-specific and general-purpose languages: comparison using a family of experiments. Empirical Software Engineering 17 (3). Pages 276-304. doi:10.1007/s10664-011-9172-x Exclusion reasons: Q1–2 [III.ajk]While this article compares languages with respect to comprehension, the methodology is such that it is unlikely that the results can provide evidence about design decision efficacy.
980. Eric Koskinen, Matthew Parkinson & Maurice Herlihy (2010): Coarse-grained transactions. In Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 19-30. doi:10.1145/1706299.1706304 Exclusion reasons: Q5 [II.ajk]Theoretical study.
981. Sirbi Kotrappa & Kulkarni Jayant Prakash (2012): The Effect of Design Patterns on Aspect Oriented Software Quality–An Empirical Evaluation. Volume 86.In Advances in Computer Science and Information Technology. Computer Science and Information Technology.Springer Berlin Heidelberg. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. Pages 357-366. doi:10.1007/978-3-642-27317-9_37 Exclusion reasons: Q1–2 [III.ajk]This article compares the Observer Pattern as implemented in Java and in AspectJ. It does not evaluate the efficacy of ny language design decision.
982. Richard J. Koubek, Gavriel Salvendy, Hubert E. Dunsmore & William K. LeBold (1989): Cognitive issues in the process of software development: review and reappraisal. International Journal of Man-Machine Studies 30 (2). Pages 171-191. doi:10.1016/S0020-7373(89)80009-4 Exclusion reasons: Q1–2 [III.ajk]This article does not review evaluations of any language design decisions.
983. M. Košık (2009): Sahara: Experimental Programming Language Based on Pict. Report. (URL unknown.) Exclusion reasons: Q5 [III.ajk]This article does not appear to exist; however, from Googling I get the impression that it was a draft of a part of Košík's PhD thesis (Matej Košík: A contribution to techniques for building dependable operating systems. PhD thesis, Slovak University of Technology, May 2011. Available at http://www2.fiit.stuba.sk/~kosik/doc/kosik-thesis.pdf), which has no aspiration to empiricity.
984. Johann M. Kraus & Hans A. Kestler (2009): Multi-core parallelization in Clojure: a case study. In Proceedings of the 6th European Lisp Workshop. New York, NY, USA: ACM. Pages 8-17. doi:10.1145/1562868.1562870 Exclusion reasons: Q5 [III.ajk]The empirical content in this article is about comparing implementations.
985. Jonathan L. Krein, Alexander C. MacLean, Charles D. Knutson, Daniel P. Delorey & Dennis L. Eggett (2010): Impact of programming language fragmentation on developer productivity: a sourceforge empirical study. International Journal of Open Source Software and Processes (IJOSSP), 2 (2). doi:10.4018/jossp.2010040104 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
986. Jonathan Leo Krein (2011): Programming Language Fragmentation and Developer Productivity: An Empirical Study. . Master's thesis at Brigham Young University. http://contentdm.lib.byu.edu/cdm4/item_viewer.php?CISOROOT=/ETD&CISOPTR=2509 Exclusion reasons: Q1–2 [III.ajk]This thesis studies the effect of using multiple languages at the same time. It does not evaluate language design decisions.
987. Wolfgang Kreutzer (1987): A Modeller's Workbench: Experiments in Object-Oriented Simulation Programming. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 203-212. doi:10.1007/3-540-47891-4_19 Exclusion reasons: Q3 [III.ajk]This article claims to describe "experiments" in object-oriented simulation programming. Assuming (with considerable doubt) that the article truly describes an empirical study, it is not discussed in enough detail and clarity to assess its validity.
988. Shriram Krishnamurthi, Matthias Felleisen & Daniel P. Friedman (1998): Synthesizing object-oriented and functional design to promote re-use. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 91-113. doi:10.1007/BFb0054088 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
989. Bent Bruun Kristensen, Ole Lehrmann Madsen, Birger Møller-Pedersen & Kristen Nygaard (1983): Abstraction mechanisms in the BETA programming language. In Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 285-298. doi:10.1145/567067.567094 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
990. Bent Bruun Kristensen, Ole Lehrmann Madsen, Birger Møller-Pedersen & Kristen Nygaard (1987): Classification of actions or Inheritance also for methods. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 98-107. doi:10.1007/3-540-47891-4_10 Exclusion reasons: Q5 [III.ajk]This analytical article has no aspiration to empiricity.
991. Bent Bruun Kristensen & Daniel C. M. May (1996): Activities: Abstractions for collective behavior. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 472-501. doi:10.1007/BFb0053074 Exclusion reasons: Q3 [III.ajk]So far as this article evaluates any language design decision empirically, this evaluation is not described in enough detail to assess its methodological quality.
992. Maxwell Krohn, Eddie Kohler & M Frans Kaashoek (2007): Events Can Make Sense. In Proc. 2007 USENIX Annual Technical Conference. Pages 87-100. http://static.usenix.org/events/usenix07/tech/krohn.html Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity regarding efficacy: there is an experience report, which is explicitly excluded by our protocol, and some implementation performance measurements.
993. Fred N. Krull (1981): Experience with ILIAD: a high-level process control language. Communications of the ACM 24 (2). Pages 66-72. doi:10.1145/358549.358555 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
994. J. Král (1986): Empirical laws of software development and their implications. Computer Physics Communications 41 (2-3). Pages 385-391. doi:10.1016/0010-4655(86)90077-9 http://www.sciencedirect.com/science/article/pii/0010465586900779 Exclusion reasons: Q1–2 [II.ajk]Programming technique discussion.
995. M. Kuittinen & J. Sajaniemi (2003): First results of an experiment on using roles of variables in teaching. In PPIG 2003. (Found in http://ppig.org/workshops/15th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]Variable roles (as used here) are not a language design issue.

996. H. E. Kulsrud (1968): Programming Languages: A general purpose graphic language. Communications of the ACM 11 (4). Pages 247-254. doi:10.1145/362991.363003 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

997. Vivek Kumar, Daniel Frampton, Stephen M. Blackburn, David Grove & Olivier Tardieu (2012): Work-stealing without the baggage. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 297-314. doi:10.1145/2384616.2384639 Exclusion reasons: Q1–2 [III.ajk]This article deals with an implementation technique, not a language design issue.

998. Reino Kurki-Suonio (1986): Towards programming with knowledge expressions. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 140-149. doi:10.1145/512644.512657 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

999. Clifton Kussmaul (2008): Novel language syntax to enhance readability: white space, parameter sets, & control structures. In OOPSLA Companion '08: Companion to the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 767-768. doi:10.1145/1449814.1449852 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1000. Maria Kutar (1999): Evaluating notations for the specification of time. In PPIG 1999. (Found in http://ppig.org/workshops/11th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article is a talk abstract regarding a then-ongoing doctoral dissertation work. That work was apparently completed in 2001 (title "Specification of temporal properties of interactive systems") and appears to be unrelated to our topic.

1001. Christian Kästner, Klaus Ostermann & Sebastian Erdweg (2012): A variability-aware module system. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 773-792. doi:10.1145/2384616.2384673 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate efficacy; it only presents feasibility. There is no meaningful comparison.

1002. Ali Sinan Köksal, Viktor Kuncak & Philippe Suter (2012): Constraints as control. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 151-164. doi:10.1145/2103656.2103675 Exclusion reasons: Q5 [III.ajk]The evaluation in this paper consists of analysis, measurements of performance mostly without comparison and personal experience reports.

1003. Thomas Kühne (1999): Internal Iteration Externalized. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 329-350. doi:10.1007/3-540-48743-3_15 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1004. Serge Lacourte (1991): Exceptions in Guide, an object-oriented language for distributed applications. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 268-287. doi:10.1007/BFb0057027 Exclusion reasons: Q1–2 [II.ajk]Abstract indicates a straightforward report of a new way to solve a language problem; there is no indication of evaluation.

1005. M. Ladkau (2007): A Wide-Spectrum Type System for Transformation Theory-Literature Review. Report. (Date based on file date on webserver (no date in document).) http://www.cse.dmu.ac.uk/STRL/research/utc/index_files/wststt_report.pdf Exclusion reasons: Q6 Q7 [III.ajk]This literature review does not aspire to empiricity.

1006. Yves Lafont (1990): Interaction nets. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 95-108. doi:10.1145/96709.96718 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1007. J.-B. Lagrange (1993): Mental Representations of String Data Types: An Experimental Study on Pupils Learning to Program. In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.

1008. James D. LAIRD, Bruce A. BURTON & Mary R. KOPPES (1986): Implementation of and Ada real-time executive: A case study. Volume 1.In NASA, Lyndon B. Johnson Space Center, First International Conference on Ada(R) Programming Language Applications for the NASA Space Station,. http://md1.csa.com/partners/viewrecord.php?requester=gs&collection=TRD&recid=N8916324AH&q=+%22programming+language%22+intitle%3A%22case+study%22&uid=788456873&setcookie=yes Exclusion reasons: Q1–2 [II.ajk]No language comparison.

1009. K. Laitinen (1993): Using Natural Naming in Programming: Feedback from Practioners. In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article discusses a choice of programming convention (namely, variable naming). It has no programming language design relevance.

1010. Wilf R. LaLonde, Dave A. Thomas & John R. Pugh (1986): An exemplar based Smalltalk. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 322-330. doi:10.1145/28697.28729 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1011. Wilf R. LaLonde & Mark Van Gulik (1988): Building a backtracking facility in smalltalk without kernel support. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 105-122. doi:10.1145/62083.62094 Exclusion reasons: Q1–2 [II.ajk]Studies whether a particular feature can be retrofitted in a particular language; does not study the efficacy of a PL DD.

1012. Wilf R. LaLonde (1989): Designing families of data types using exemplars. ACM Transactions on Programming Languages and Systems 11 (2). Pages 212-248. doi:10.1145/63264.63265 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1013. Patrick Lam & Martin Rinard (2003): A Type System and Analysis for the Automatic Extraction and Enforcement of Design Information. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 275-302. doi:10.1007/978-3-540-45070-2_13 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1014. Gloria J. Lambert (1973): Large scale file processing: POGOL. In Proc. 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 226-234. doi:10.1145/512927.512948 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.

1015. Butler W. Lampson & Eric E. Schmidt (1983): Practical use of a polymorphic applicative language. In Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 237-255. doi:10.1145/567067.567070 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1016. T. K. Landauer, K. M. Galotti & S. Hartwell (1983): Natural command names and initial learning: a study of text-editing terms. Communications of the ACM 26 (7). Pages 495-503. doi:10.1145/358150.358157 Exclusion reasons: Q1–2 [III.ajk]This article reports several empirical studies related to the design of command-based user interfaces. It has no real relevance to programming language design.

1017. P. J. Landin (1965): Correspondence between ALGOL 60 and Church's Lambda-notation: part I. Communications of the ACM 8 (2). Pages 89-101. doi:10.1145/363744.363749 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1018. P. J. Landin (1965): A correspondence between ALGOL 60 and Church's Lambda-notations: Part II. Communications of the ACM 8 (3). Pages 158-167. doi:10.1145/363791.363804 Exclusion reasons: Q5 [III.ajk]This is a formal, theoretical study on the correspondence of Algol and lambda calculus. There is no empirical content.

1019. P. J. Landin (1966): The next 700 programming languages. Communications of the ACM 9 (3). Pages 157-166. doi:10.1145/365230.365257 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1020. Carl E. Landwehr (1980): An Abstract Type for Statistics Collection in Simula. ACM Transactions on Programming Languages and Systems 2 (4). Pages 544-563. doi:10.1145/357114.357118 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1021. Kevin J. Lang & Barak A. Pearlmutter (1986): Oaklisp: an object-oriented scheme with first class types. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 30-37. doi:10.1145/28697.28701 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1022. Jun Lang & David B. Stewart (1998): A study of the applicability of existing exception-handling techniques to component-based real-time software technology. ACM Transactions on Programming Languages and Systems 20 (2). Pages 274-301. doi:10.1145/276393.276395 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.

1023. Hans Langtangen (2007): A Case Study in High-Performance Mixed-Language Programming. Volume 4699.In Kågström, Bo and Elmroth, Erik and Dongarra, Jack and Wasniewski, Jerzy (ed.) Applied Parallel Computing. State of the Art in Scientific Computing.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 36-49. doi:10.1007/978-3-540-75755-9_4 Exclusion reasons: Q5 [III.ajk]This article analytically compares several languages; it also has a performance evaluation section that has no relevance to language design.

1024. Hans Petter Langtangen & Xing Cai (2008): On the Efficiency of Python for High-Performance Computing: A Case Study Involving Stencil Updates for Partial Differential Equations. In Bock, Hans Georg and Kostina, Ekaterina and Phu, Hoang Xuan and Rannacher, Rolf (ed.) Modeling, Simulation and Optimization of Complex Processes.Springer Berlin Heidelberg. Pages 337-357. doi:10.1007/978-3-540-79409-7_23 Exclusion reasons: Q1–2 [III.ajk]This article studies the performance impact of using Python for numerical high-performance computing. Although it compares Python to other languages, it does this in a way that doesn't evaluate the language design; rather it is evaluating the implementations and platforms involved in the study.

1025. James R. Larus (1993): Compiling for shared-memory and message-passing computers. ACM Transactions on Programming Languages and Systems 2 (1-4). Pages 165-180. doi:10.1145/176454.176514 Exclusion reasons: Q1–2 [III.ajk]This article discusses implementation techniques and is not relevant to PL design.

1026. Mark Lattanzi & Sallie Henry (1998): Software reuse using C++ classes: The question of inheritance. Journal of Systems and Software 41 (2). Pages 127-132. doi:10.1016/S0164-1212(97)10013-9 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1027. Jane Laursen & Robert Atkinson (1987): Opus: A Smalltalk production system. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 377-387. doi:10.1145/38765.38841 Exclusion reasons: Q1–2 [II.ajk]System exposition.

1028. Jari M. Lavonen, Veijo P. Meisalo, Matti Lattu & Erkki Sutinen (2003): Concretising the programming task: a case study in a secondary school. Computers & Education 40 (2). Pages 115-135. (.) doi:10.1016/S0360-1315(02)00101-X Exclusion reasons: Q1–2 [II.ajk]Studies teaching, and visual languages aren't PLs by our definition.

1029. D. H. Lawrie, T. Layman, D. Baer & J. M. Randal (1975): Glypnir - a programming language for Illiac IV. Communications of the ACM 18 (3). Pages 157-164. doi:10.1145/360680.360687 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.
1030. Harold W. Lawson, Jr. (1967): PL/I list processing. Communications of the ACM 10 (6). Pages 358-367. doi:10.1145/363332.363344 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1031. B. M. Leavenworth (1966): Syntax macros and extended translation. Communications of the ACM 9 (11). Pages 790-793. doi:10.1145/365876.365879 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1032. Ulrike Lechner, Christian Lengauer, Friederike Nickl & Martin Wirsing (1996): (Objects + concurrency) & reusability — A proposal to circumvent the inheritance anomaly. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 232-247. doi:10.1007/BFb0053064 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1033. François Leclerc & Christine Paulin-Mohring (1994): Programming with streams in Coq a case study: The Sieve of Eratosthenes. Volume 806.In Barendregt, Henk and Nipkow, Tobias (ed.) Types for Proofs and Programs.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 191-212. doi:10.1007/3-540-58085-9_77 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work.
1034. Henry F. Ledgard & Michael Marcotty (1975): A genealogy of control structures. Communications of the ACM 18 (11). Pages 629-639. doi:10.1145/361219.361222 Exclusion reasons: Q6 Q7 [II.ajk]Review of theoretical studies.
1035. Henry F. Ledgard & William C. Cave (1976): Cobol under control. Communications of the ACM 19 (11). Pages 601-608. doi:10.1145/360363.360366 Exclusion reasons: Q1–2 [II.ajk]Coding standard exposition.
1036. Henry Ledgard, John A. Whiteside, Andrew Singer & William Seymour (1980): The natural language of interactive systems. Communications of the ACM 23 (10). Pages 556-563. doi:10.1145/359015.359018 Exclusion reasons: Q1–2 [III.ajk]This article presents a study evaluating interactive command languages. It is not relevant to PL design.
1037. Henry F. Ledgard & Andrew Singer (1982): Scaling down Ada (or towards a standard Ada subset). Communications of the ACM 25 (2). Pages 121-125. doi:10.1145/358396.358402 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiicity.
1038. Pascal Ledru (1997): Adaptive parallelism: an early experiment with Java remote method invocation. SIGOPS Operating Systems Review 31. Pages 24-29. doi:10.1145/271019.271024 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
1039. Jeng Kuen Lee & D. Gannon (1991): Object oriented parallel programming: experiments and results. In Supercomputing, 1991. Supercomputing '91. Proceedings of the 1991 ACM/IEEE Conference on. Pages 273-282. doi:10.1145/125826.105186 Exclusion reasons: Q1–2 [III.ajk]This article does not compare its language design to another.
1040. Shinn-Der Lee & Daniel P. Friedman (1993): Quasi-static scoping: sharing variable bindings across multiple lexical scopes. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 479-372. doi:10.1145/158511.158706 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1041. Adrienne Lee & Nancy Pennington (1994): The effects of paradigm on cognitive activities in design. International Journal of Human-Computer Studies 40 (4). Pages 577-601. doi:10.1006/ijhc.1994.1028 Exclusion reasons: Q1–2 [II.ajk]Design, not programming.
1042. Keunwoo Lee & Craig Chambers (2006): Parameterized Modules for Classes and Extensible Functions. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 353-378. doi:10.1007/11785477_21 Exclusion reasons: Q1–2 [II.ajk]Language exposition only, based on the abstract.
1043. Jonathan Lee, Jens Palsberg, Rupak Majumdar & Hong Hong (2012): Efficient May Happen in Parallel Analysis for Async-Finish Parallelism. Volume 7460.In Miné, Antoine and Schmidt, David (ed.) Static Analysis.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 5-23. doi:10.1007/978-3-642-33125-1_4 Exclusion reasons: Q1–2 [II.ajk]This article studies an implementation technique, not any language design issues.
1044. Byeongcheol Lee, Robert Grimm, Martin Hirzel & Kathryn McKinley (2012): Marco: Safe, Expressive Macros for Any Language. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 589-613. doi:10.1007/978-3-642-31057-7_26 Exclusion reasons: Q5 [II.ajk]No empirical evaluation.
1045. George B. Leeman, Jr. (1986): A formal approach to undo operations in programming languages. ACM Transactions on Programming Languages and Systems 8 (1). Pages 50-87. doi:10.1145/5001.5005 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1046. Avraham Leff & James T. Rayfield (2007): Webrb: evaluating a visual domain-specific language for building relational web-applications. In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. Pages 281-300. doi:10.1145/1297027.1297048 Exclusion reasons: Q1–2 [III.ajk]This article concerns a visual language; such languages are excluded by our definition.
1047. J. A. Lehman (1989): An empirical comparison of textual and graphical data structure documentation for Cobol programs. Software Engineering, IEEE Transactions on 15 (9). Pages 1131-1135. doi:10.1109/32.31370 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
1048. Torsten Leidig & Peter Roesch (1994): EXL: An Experimental Lisp Dialect. Paper in Citeseer. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.24.6760 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1049. Daan Leijen (2009): Flexible types: robust type inference for first-class polymorphism. In Proc. 36th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 66-77. doi:10.1145/1480881.1480891 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1050. K. Rustan M. Leino & Greg Nelson (2002): Data abstraction and information hiding. ACM Transactions on Programming Languages and Systems 24 (5). Pages 491-553. doi:10.1145/570886.570888 Exclusion reasons: Q1–2 [II.ajk]Studies formal verification, no PL design relevance.
1051. K. Rustan M. Leino & Peter Müller (2004): Object Invariants in Dynamic Contexts. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 491-515. doi:10.1007/978-3-540-24851-4_22 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1052. Ronald S. Lemos (1979): An implementation of structured walk-throughs in teaching Cobol programming. Communications of the ACM 22 (6). Pages 335-340. doi:10.1145/359114.359116 Exclusion reasons: Q1–2 [II.ajk]Study of a teaching method.
1053. Benjamin S. Lerner, Herman Venter & Dan Grossman (2010): Supporting dynamic, third-party code customizations in JavaScript using aspects. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications. Pages 361-376. doi:10.1145/1869459.1869490 Exclusion reasons: Q1–2Q5 [III.ajk]This article does not evaluate the efficacy of its design decisions – the only evaluation presented pertains to implementation efficiency and the adequacy of the constructs for the planned purpose.
1054. Xavier Leroy (1993): Polymorphism by name for references and continuations. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 220-231. doi:10.1145/158511.158632 Exclusion reasons: Q3 Disagreement resolution result. [III.ajk]This article describes and analyzes a new approach to polymorphism for references and continuations. There is a brief empirical note in the article, but it is not detailed enough to assess its quality. There is a fuller treatment in the topic in Leroy's dissertation (reference 11 in the paper), but it is in French and thus excluded.
1055. Xavier Leroy (1994): Manifest types, modules, and separate compilation. In Proc. 21th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 109-122. doi:10.1145/174675.176926 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1056. Xavier Leroy (1995): Applicative functors and fully transparent higher-order modules. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 142-153. doi:10.1145/199448.199476 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1057. Julian C. Leslie (1981): State notation programming languages in psychology. International Journal of Man-Machine Studies 14 (3). Pages 341-354. doi:10.1016/S0020-7373(81)80062-4 Exclusion reasons: Q5 [II.ajk]No empirical aspect.
1058. Yves Lespérance, Hector Levesque & Shane Ruman (1997): An experiment in using Golog to build a personal banking assistant. Volume 1209.In Cavedon, Lawrence and Rao, Anand and Wobcke, Wayne (ed.) Intelligent Agent Systems Theoretical and Practical Issues. Pages 27-43. doi:10.1007/3-540-62686-7_26 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.
1059. Jacek Leszczylowski (1980): An experiment with "Edinburgh LCF". Volume 87.In Bibel, Wolfgang and Kowalski, Robert (ed.) 5th Conference on Automated Deduction Les Arcs, France, July 8–11, 1980. Pages 170-181. doi:10.1007/3-540-10009-1_14 Exclusion reasons: Q5 [III.ajk]This article does not, despite its title, aspire to empiricity.
1060. Charl de Leur (2009): Evaluation of Multi-Core Programming Models. In 11th Twente Student Conference on IT. http://fmt.cs.utwente.nl/files/sprojects/78.pdf Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity: the comparison is analytical with respect to efficacy concerns.
1061. Laura Marie Leventhal (1988): Experience of programming beauty: some patterns of programming aesthetics. International Journal of Man-Machine Studies 28 (5). Pages 525-550. doi:10.1016/S0020-7373(88)80059-2 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1062. C. H. Lewis & B. K. Rosen (1973): Recursively defined data types: part 1. In Proc. 1st Annual ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 125-138. doi:10.1145/512927.512939 Exclusion reasons: Q5 [III.ajk]This formal article does not aspire to empiricity.

1063. J. A. Lewis (1989): A Controlled Experiment to Identify Factors Affecting Software Reuse. In Proceedings of the 19th Annual Virginia Computer Users Conference. Exclusion reasons: Q3 [III.ajk]Worlcat reveals that only Viginia Tech library holds this proceedings series; looking it up on its own records, its oldest volume is 20th (1990). The paper is also not online.

1064. Clayton Lewis (1992): Addressing the psychology of programming in programming language design. In PPIG 1992. (Found in http://ppig.org/workshops/4th-programme.html.) Exclusion reasons: Q3 [III.ajk]I cannot find any trace of this proceedings book, and http://spot.colorado.edu/~clayton/vita.html makes clear that this was not a publication as the author counts things. Thus I consider asking for an interlibrary loan a waste of resources and exclude this summarily.

1065. Jeffrey R. Lewis, John Launchbury, Erik Meijer & Mark B. Shields (2000): Implicit parameters: dynamic scoping with static types. In Proc. 27th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 108-118. doi:10.1145/325694.325708 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1066. Ondřej Lhoták & Laurie Hendren (2008): Relations as an abstraction for BDD-based program analysis. ACM Transactions on Programming Languages and Systems 30 (4). doi:10.1145/1377492.1377494 Exclusion reasons: Q1–2 [II.ajk]Language exposition, implementation issues discussion.

1067. Wing Ning Li & Ravi Kiran (1996): An object-oriented design and implementation of reusable graph objects with C++: a case study. In Proceedings of the 1996 ACM symposium on Applied Computing. New York, NY, USA: ACM. Pages 510-514. doi:10.1145/331119.331433 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions, at least not in a comparative sense.

1068. Wenlong Li, E. Li, Ran Meng, Tao Wang & C. Dulong (2006): Performance analysis of Java concurrent programming: a case study of video mining system. In Parallel and Distributed Processing Symposium, 2006. IPDPS 2006. 20th International. Pages 8 pp.. doi:10.1109/IPDPS.2006.1639505 Exclusion reasons: Q1–2 [II.ajk]Language evaluation without a comparison.

1069. Peng Li & Steve Zdancewic (2007): Combining events and threads for scalable network services implementation and evaluation of monadic, application-level concurrency primitives. In Proceedings of the 2007 ACM SIGPLAN conference on Programming language design and implementation. New York, NY, USA: ACM. PLDI '07. Pages 189-199. doi:10.1145/1250734.1250756 Exclusion reasons: Q1–2 [II.ajk]The empirical content of this article merely evaluates implementation efficiency.

1070. Zhen Li, Zhe Zhao & Eileen Kraemer (2010): Characterizing Comprehension of Concurrency Concepts. In PPIG 2010. (Found in http://ppig2010.org/index.php?title=Program.) Exclusion reasons: Q1–2 [III.ajk]Paper found at http://ppig.org/papers/22nd-Teach-1.pdf. This article does not evaluate a language design decision.

1071. Guodong Li, Robert Palmer, Michael DeLisi, Ganesh Gopalakrishnan & Robert M. Kirby (2011): Formal specification of MPI 2.0: Case study in specifying a practical concurrent programming API. Science of Computer Programming 76 (2). Pages 65-81. doi:10.1016/j.scico.2010.03.007 Exclusion reasons: Q1–2 [II.ajk]Formal specification, not programming, language.

1072. Siliang Li & Gang0 Tan (2011): JET: exception checking in the Java native interface. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 345-358. doi:10.1145/2048066.2048095 Exclusion reasons: Q1–2 [II.ajk]Implementation technique at best.

1073. Du Li, Witawas Srisa-an & Matthew B. Dwyer (2011): SOS: saving time in dynamic race detection with stationary analysis. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 35-50. doi:10.1145/2048066.2048072 Exclusion reasons: Q1–2 [II.ajk]This article deals with a diagnostic technique only.

1074. Sheng Liang, Paul Hudak & Mark Jones (1995): Monad transformers and modular interpreters. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 333-343. doi:10.1145/199448.199528 Exclusion reasons: Q1–2 [III.ajk]Programming technique development; no comparison language (beyond the trivial Gofer/Haskell comparison).

1075. Y. K. C. Liao & G. W. Bright (1991): Effects of computer programming on cognitive outcomes: A meta-analysis. Journal of Educational Computing Research 7 (3). Pages 251-266. doi:10.2190/E53G-HH8K-AJRR-K69M Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1076. Karl Lieberherr, David H. Lorenz & Pengcheng Wu (2003): A case for statically executable advice: checking the law of demeter with AspectJ. In Proceedings of the 2nd international conference on Aspect-oriented software development. New York, NY, USA: ACM. AOSD '03. Pages 40–49. doi:10.1145/643603.643608 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1077. Karl Lieberherr, Boaz Patt-Shamir & Doug Orleans (2004): Traversals of object structures: Specification and Efficient Implementation. ACM Transactions on Programming Languages and Systems 26 (2). Pages 370-412. doi:10.1145/973097.973102 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity. The section that claims to present empirical evidence merely provides anecdotes.

1078. Bennet P. Lientz (1976): A comparative evaluation of versions of BASIC. Communications of the ACM 19 (4). Pages 175-181. doi:10.1145/360032.360038 Exclusion reasons: Q1–2 [III.ajk]This article reports a comparison of various implementations of BASIC, based on both feature availability and computational performance. It does not evaluate a PL design decision.

1079. P.H. Lim (1989): A Comparative Case Study of Programming Language Expansion Ratios. . MSc at Massey University. (No URL known.) Exclusion reasons: Q1–2 [III.ajk]This thesis evaluates a system size estimation tool, not any language design decisions.

1080. Chuan-kai Lin & Andrew P. Black (2007): DirectFlow: A Domain-Specific Language for Information-Flow Systems. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 299-322. doi:10.1007/978-3-540-73589-2_15 Exclusion reasons: Q1–2 [II.ajk]Language and system exposition.

1081. Daniel Lincke & Sibylle Schupp (2009): The function concept in C++: an empirical study. In Proceedings of the 2009 ACM SIGPLAN workshop on Generic programming. Pages 25-36. doi:10.1145/1596614.1596619 Exclusion reasons: Q1–2 [III.ajk]This article compares several ways to simulate first-class functions in C++. It does not evaluate a language design decision.

1082. Gary Lindstrom (1978): Control structure aptness: A case study using top-down parsing. In Proceedings of the 3rd international conference on Software engineering. Piscataway, NJ, USA: IEEE Press. Pages 5-12. http://dl.acm.org/citation.cfm?id=800099.803184 Exclusion reasons: Q5 [III.ajk]This article presents an evaluation of various control structures from the point of view of implementing backtracking searches. However, the approach is clearly analytic, not empirical.

1083. Gary Lindstrom (1979): Backtracking in a Generalized Control Setting. ACM Transactions on Programming Languages and Systems 1 (1). Pages 8-26. doi:10.1145/357062.357063 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1084. Gary Lindstrom & Mary Lou Soffa (1981): Referencing and Retention in Block-Structured Coroutines. ACM Transactions on Programming Languages and Systems 3 (3). Pages 263-292. doi:10.1145/357139.357143 Exclusion reasons: Q5 [III.ajk]This article is purely mathematical and has no aspirations to empiricity.

1085. Gary Lindstrom (1985): Functional programing and the logical variable. In Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 266-280. doi:10.1145/318593.318657 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article does not aspire to empiricity.

1086. Martin Lippert & Cristina Videira Lopes (2000): A study on exception detection and handling using aspect-oriented programming. In Proceedings of the 22nd international conference on Software engineering. ICSE '00. Pages 418-427. doi:10.1145/337180.337229 Exclusion reasons: Q5 [III.ajk]This article is largely analytical. The quantitative results are presented almost as an afterthought and it is not clear how they are influenced by other matters than the language design decision that possibly is at issue.

1087. Richard J. Lipton, Robert Sedgewick & Jacobo Valdes (1982): Programming aspects of VLSI (preliminary version). In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 57-65. doi:10.1145/582153.582160 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1088. Luigi Liquori & Arnaud Spiwack (2008): FeatherTrait: A modest extension of Featherweight Java. ACM Transactions on Programming Languages and Systems 30 (2). doi:10.1145/1330017.1330022 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical study.

1089. Barbara Liskov, Alan Snyder, Russell Atkinson & Craig Schaffert (1977): Abstraction mechanisms in CLU. Communications of the ACM 20 (8). Pages 564-573. doi:10.1145/359763.359789 Exclusion reasons: Q1–2 [II.ajk]Discusses the usefulness of a single language; there is no comparison so far as the abstract is accurate.

1090. Barbara Liskov & Robert Scheifler (1982): Guardians and actions: linguistic support for robust, distributed programs. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 7-19. doi:10.1145/582153.582155 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.

1091. Barbara Liskov & Robert Scheifler (1983): Guardians and Actions: Linguistic Support for Robust, Distributed Programs. ACM Transactions on Programming Languages and Systems 5 (3). Pages 371-404. doi:10.1145/2166.357215 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1092. Barbara Liskov, Maurice Herlihy & Lucy Gilbert (1986): Limitations of synchronous communication with static process structure in languages for distributed computing. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 150-159. doi:10.1145/512644.512658 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1093. Barbara Liskov (1988): Distributed programming in Argus. Communications of the ACM 31 (3). Pages 300-312. doi:10.1145/42392.42399 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.

1094. Barbara H. Liskov & Jeannette M. Wing (1994): A behavioral notion of subtyping. ACM Transactions on Programming Languages and Systems 16 (6). Pages 1811-1841. doi:10.1145/197320.197383 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1095. Raymond Lister, Anders Berglund, Tony Clear, Joe Bergin, Kathy Garvin-Doxas, Brian Hanks, Lew Hitchner, Andrew Luxton-Reilly, Kate Sanders, Carsten Schulte & Jacqueline L. Whalley (2006): Research perspectives on the objects-early debate. In Working group reports on ITiCSE on Innovation and technology in computer science education. New York, NY, USA: ACM. ITiCSE-WGR '06. Pages 146-165. doi:10.1145/1189215.1189183 Exclusion reasons: Q1–2 [II.ajk]This article studies a discussion regarding teaching approach and does not evaluate any language design decisions.

1096. Charles R. Litecky & Gordon B. Davis (1976): A study of errors, error-proneness, and error diagnosis in Cobol. Communications of the ACM 19 (1). Pages 33-38. doi:10.1145/359970.359991 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1097. Ying Chun Liu (2002): Comparison between C++ and Java: a case study using a networked automated gas station stimulation system . . MSc at Concordia University. http://spectrum.library.concordia.ca/1655/ Exclusion reasons: Q5 [III.ajk]This master's thesis evaluates comparatively C++ and Java by having the author implement the same system in both languages and then comparing the languages, both analytically and in light of that development experience. While the implementation is perhaps empirical in nature, the conclusions are mainly drawn from the analytical comparison. Thus this thesis does not in any meaningful sense offer empirical evidence about the comparison.

1098. Yu David Liu & Scott F. Smith (2004): Modules with Interfaces for Dynamic Linking and Communication. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 415-439. doi:10.1007/978-3-540-24851-4_19 Exclusion reasons: Q5 [III.ajk]This construcive-analytical study has no aspiration for empiricity.

1099. Yanhong A. Liu, Scott D. Stoller, Michael Gorbovitski, Tom Rothamel & Yanni Ellen Liu (2005): Incrementalization across object abstraction. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 473-486. doi:10.1145/1094811.1094848 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.

1100. Feng Liu, O. A. Mohamed, Xiaoyu Song & Qingping Tan (2009): A case study on system-level modeling by aspect-oriented programming. In Quality of Electronic Design, 2009. ISQED 2009. Quality Electronic Design. Pages 345-349. doi:10.1109/ISQED.2009.4810318 Exclusion reasons: Q5 [III.ajk]This article expores the implications of the two language designs and are thus analytic, not empirical under our definition.

1101. Tongping Liu & Emery D. Berger (2011): SHERIFF: precise detection and automatic mitigation of false sharing. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 3-18. doi:10.1145/2048066.2048070 Exclusion reasons: Q1–2 [III.ajk]Sheriff is a runtime instrumentation tool for detecting or protecting from certain implementation artefacts that cause performance loss. There is no language design issue at stake.

1102. Yanhong A. Liu, Scott D. Stoller, Bo Lin & Michael Gorbovitski (2012): From clarity to efficiency for distributed algorithms. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 395-410. doi:10.1145/2384616.2384645 Exclusion reasons: Q5 [III.ajk]The evaluation is analytical in nature (even the quantitative evaluations are measures of how well they can be used [as proxied by the authors' best effort and efforts in the literature], not how well they are actually used by programmers).

1103. Lionello A. Lombardi (1964): A general business-oriented language based on decision expressions. Communications of the ACM 7 (2). Pages 104-111. doi:10.1145/363921.363939 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1104. Yuheng Long, Hridesh Rajan & Sean L. Mooney (2010): Reconciling concurrency and modularity with Panini's asynchronous typed events. In SPLASH '10 Systems Programming Languages and Applications: Software for Humanity. Pages 243-244. doi:10.1145/1869542.1869595 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1105. Alex Loopik & Yossi Lichtenstein (1992): Prolog versus Kee: A case study. Artificial Intelligence in Engineering 7 (3). Pages 153-165. doi:10.1016/0954-1810(92)90003-K Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which the same program specification is implemented in two different languages and the different choices made in each program are then described. The study appears to be relevant but does not announce any efficacy conclusions, nor are they evident from the discussion.

1106. Cristina Videira Lopes & Karl J. Lieberherr (1994): Abstracting process-to-function relations in concurrent object-oriented applications. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 81-99. doi:10.1007/BFb0052177 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1107. Gus Lopez, Bjørn Freeman-Benson & Alan Borning (1994): Constraints and object identity. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 260-279. doi:10.1007/BFb0052187 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1108. Roberto E. Lopez-Herrejon, Don Batory & William Cook (2005): Evaluating Support for Features in Advanced Modularization Technologies. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 169-194. doi:10.1007/11531142_8 Exclusion reasons: Q5 [III.ajk]This article compares analytically a number of aspect languages. It does not aspire to empiricity.

1109. Roberto Lopez-Herrejon & Don Batory (2007): Modeling Features in Aspect-Based Product Lines with Use Case Slices: An Exploratory Case Study. Volume 4364.In Kühne, Thomas (ed.) Models in Software Engineering.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 6-16. doi:10.1007/978-3-540-69489-2_2 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1110. David H. Lorenz & Boaz Rosenan (2011): Cedalion: a language for language oriented programming. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 733-752. doi:10.1145/2048066.2048123 Exclusion reasons: Q5 [III.ajk]This article includes a participant-observer case study in which the new language was used to reimplement a program for a real user organization; however, it merely demonstrates the language being able to deliver, not whether it performs better than another choice (and it could arguably qualify under the experience report exclusion criterion).

1111. David H. Lorenz & Boaz Rosenan (2011): A case study of language oriented programming with cedalion: [extended abstract]. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 199-200. doi:10.1145/2048147.2048205 Exclusion reasons: Q4 Q5 [III.ajk]This extended abstract is not a complete report; I was unable to find one. It is also unlikely that any full report would be included, as the evaluation looks analytic.

1112. D. H. Lorenz & B. Rosenan (2011): A Comparative Case Study of Code Reuse With Language Oriented Programming. paper in arXiv. http://adsabs.harvard.edu/abs/2011arXiv1103.5901L Exclusion reasons: Q5 [III.ajk]The evaluation in this article is analytical in nature.

1113. Ronald P. Loui (2008): In Praise of Scripting: Real Programming Pragmatism. Computer 41 (7). Pages 22-26. doi:10.1109/MC.2008.228 Exclusion reasons: Q1–2 [III.ajk]This article does not present or summarise any evaluative studies.

1114. Tom Love (1977): An experimental investigation of the effect of program structure on program understanding. In Proceedings of an ACM conference on Language design for reliable software. New York, NY, USA: ACM. Pages 105-113. doi:10.1145/800022.808317 Exclusion reasons: Q1–2 [III.ajk]This article reports a controlled experiment with human participants in which the participants were asked to memorise and then recall programs. The programs in play were picked from a published book and then modified in several systematic ways to produce variants differing in the independent variable (presence or absence of structured programming, and presence or absence of paragraphing). Memorization-and-recall was used as a proxy of program understanding. Structured programming was defined as the use of sequencing, selection and repetition and the absence of middle exit and goto. While language design can be influenced by the principles of structured programming, this experiment does not evaluate any particular design decisions (it only evaluates programming style).

1115. David W. Low (1973): Programming by questionnaire: an effective way to use decision tables. Communications of the ACM 16 (5). Pages 282-286. doi:10.1145/362041.362194 Exclusion reasons: Q1–2 [II.ajk]Language (or at least programming approach) exposition, with no comparison.

1116. James R. Low (1978): Automatic data structure selection: an example and overview. Communications of the ACM 21 (5). Pages 376-385. doi:10.1145/359488.359498 Exclusion reasons: Q1–2 [II.ajk]Implementation technique discussion.

1117. G. Low & S. Huan (1999): Impact of object oriented development on software quality. In Software Technology and Engineering Practice, 1999. STEP '99. Proceedings. Pages 3-11. doi:10.1109/STEP.1999.798402 Exclusion reasons: Q1–2 [II.ajk]This article evaluates the efficacy of CASE tools, not any language design decision.

1118. David Lowe & John Leaney (1993): Has the Pascal Experiment Failed? or Can A Good Language make Good Programmers?. In Seventh Australian Software Engineering Conference, Sydney. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.49.6336 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decision, rather it evaluates a course using a language. It is not of any relevance to language design.

1119. P. Geoffrey Lowney (1981): Carrier arrays: an idiom-preserving extension to APL. In Proc. 8th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 1-13. doi:10.1145/567532.567533 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1120. R. E. Lowrance (2009): APL Literature Review. Report. http://www.cs.nyu.edu/manycores/litrev.pdf Exclusion reasons: Q6 Q7 [III.ajk]This work appears to be a draft of a narrative review of APL literature. It focuses on language design and usage, as well as implementation issues. The bibliography does not include primary or secondary studies that appear to evaluate the efficacy of a PL design decision.

1121. Yi Lu & John Potter (2005): A Type System for Reachability and Acyclicity. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 479-503. doi:10.1007/11531142_21 Exclusion reasons: Q5 [III.ajk]This analytical-constructive study does not aspire to empiricity.

1122. Yi Lu & John Potter (2006): Protecting representation with effect encapsulation. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 359-371. doi:10.1145/1111037.1111069 Exclusion reasons: Q5 [II.ajk]Type-theoretic development

1123. Yi Lu & John Potter (2006): On Ownership and Accessibility. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 99-123. doi:10.1007/11785477_6 Exclusion reasons: Q5 [II.ajk]Type-theoretic study.

1124. Yi Lu, John Potter & Jingling Xue (2007): Validity Invariants and Effects. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 202-226. doi:10.1007/978-3-540-73589-2_11 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.

1125. Hongmin Lu, Yuming Zhou, Baowen Xu, Hareton Leung & Lin Chen (2012): The ability of object-oriented metrics to predict change-proneness: a meta-analysis. Empirical Software Engineering 17 (3). Pages 200-242. doi:10.1007/s10664-011-9170-z Exclusion reasons: Q1–2 [II.ajk]This article deals with metrics only.

1126. Roberto Lublinerman, Swarat Chaudhuri & Pavol Cerny (2009): Parallel programming with object assemblies. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications. Pages 61-80. doi:10.1145/1640089.1640095 Exclusion reasons: Q1–2 [III.ajk]This article contains a set of self-styled case studies, which are merely examples and thus not empirical; it also contains a performance evaluation using some of those examples. The evaluation appears to be mostly focuising on the implementation and not on the language design.

1127. Roberto Lublinerman, Jisheng Zhao, Zoran Budimlić, Swarat Chaudhuri & Vivek Sarkar (2011): Delegated isolation. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 885-902. doi:10.1145/2048066.2048133 Exclusion reasons: Q1–2 [III.ajk]The evaluation is based on benchmarks instead of actual applications; the measurements are about performance. It is hard to see any insight this gives on efficacy as opposed to just implementation skill.

1128. J. M. Lucassen & D. K. Gifford (1988): Polymorphic effect systems. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 47-57. doi:10.1145/73560.73564 Exclusion reasons: Q3 [III.ajk]There arguably is an empirical evaluation of efficacy of effect masking, but it is a mere statement of results, with close to no description of methodology.

1129. Steven Lucco & Oliver Sharp (1991): Parallel programming with coordination structures. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 197-208. doi:10.1145/99583.99612 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1130. C.J.P. Lucena & T.H.C. Pequeno (1979): Program Derivation Using Data Types: A Case Study. Software Engineering, IEEE Transactions on SE-5 (6). Pages 586-592. doi:10.1109/TSE.1979.230194 Exclusion reasons: Q1–2 [II.ajk]No PL design relevance.

1131. David C. Luckham & Norihisa Suzuki (1979): Verification of Array, Record, and Pointer Operations in Pascal. ACM Transactions on Programming Languages and Systems 1 (2). Pages 226-244. doi:10.1145/357073.357078 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1132. David C. Luckham & W. Polak (1980): Ada exception handling: an axiomatic approach. ACM Transactions on Programming Languages and Systems 2 (2). Pages 225-233. doi:10.1145/357094.357100 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1133. PAUL LUFF & CHRISTIAN HEATH (2000): The collaborative production of computer commands in command and control. International Journal of Human-Computer Studies 52 (4). Pages 669-699. doi:10.1006/ijhc.1999.0354 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate language design decisions.

1134. S.M. Luk (1989): An experimental visual programming language. Brigham Young University. Dept. of Computer Science.. (No URL known.) Exclusion reasons: Q1–2 [II.ajk]Visual languages aren't programming languages by our definition.

1135. Ralf Lämmel (2007): Scrap your boilerplate with XPath-like combinators. In Proc. 34th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 137-142. doi:10.1145/1190216.1190240 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity,

1136. Konstantin Läufer & Martin Odersky (1994): Polymorphic type inference and abstract data types. ACM Transactions on Programming Languages and Systems 16 (5). Pages 1411-1430. doi:10.1145/186025.186031 Exclusion reasons: Q5 [II.ajk]Formal type-theoretical work.

1137. Klaus-Peter Löhr (1992): Concurrency annotations. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications. Pages 327-340. doi:10.1145/141936.141964 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1138. QingMing Ma (1992): Parametricity as subtyping. In Proc. 19th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 281-292. doi:10.1145/143165.143225 Exclusion reasons: Q5 [III.ajk]This is formal, theoretical work with no aspirations of empiricality.

1139. Steve MacDonald, Kai Tan, Jonathan Schaeffer & Duane Szafron (2009): Deferring design pattern decisions and automating structural pattern changes using a design-pattern-based programming system. ACM Transactions on Programming Languages and Systems 31 (3). doi:10.1145/1498926.1498927 Exclusion reasons: Q1–2 [II.ajk]No PL design issues.

1140. I. Macia, A. Garcia, A. von Staa, J. Garcia & N. Medvidovic (2011): On the Impact of Aspect-Oriented Code Smells on Architecture Modularity: An Exploratory Study. In Software Components, Architectures and Reuse (SBCARS), 2011 Fifth Brazilian Symposium on. Pages 41 -50. doi:10.1109/SBCARS.2011.18 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1141. Isela Macia Bertran, Alessandro Garcia & Arndt von Staa (2011): An exploratory study of code smells in evolving aspect-oriented systems. In Proceedings of the tenth international conference on Aspect-oriented software development. New York, NY, USA: ACM. Pages 203-214. doi:10.1145/1960275.1960300 Exclusion reasons: Q1–2 [II.ajk]There does not seem to be any programming language design decision under evaluation.

1142. R. I. Mackie & R. R. Gajewski (1998): Object Oriented Programming and Finite Element Analysis: Achieving Control Over the Calculation Process. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 580. doi:10.1007/3-540-49255-0_147 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.

1143. R. I. Mackie (2001): Object oriented programming for structural mechanics: a review. In Topping, B. H. V. (ed.) Civil and structural engineering computing: 2001.Saxe-Coburg Publications. Pages 137-159. http://dl.acm.org/citation.cfm?id=771946.771953 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1144. Thomas N. Mackinson (1961): COBOL: a sample problem. Communications of the ACM 4 (8). Pages 340-346. doi:10.1145/366678.366687 Exclusion reasons: Q1–2 [III.ajk]This paper does not report a study.

1145. Matthew B. MacLaurin (2011): The design of kodu: a tiny visual programming language for children on the Xbox 360. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 241-246. doi:10.1145/1926385.1926413 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1146. David B. MacQueen (1986): Using dependent types to express modular structure. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 277-286. doi:10.1145/512644.512670 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1147. Ravichandhran Madhavan & Raghavan Komondoor (2011): Null dereference verification via over-approximated weakest pre-conditions analysis. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 1033-1052. doi:10.1145/2048066.2048144 Exclusion reasons: Q1–2 [III.ajk]The evaluation in this paper does not use any comparison technology and there is no evaluation of a design decision. The related work section does compare the evaluation results in this paper to evaluation results in reported in related research papers, but as they note themselves, this comparison is not very compelling as the evaluation methodologies differ.

1148. Nazim H. Madhavji (1984): Visibility aspects of programmed dynamic data structures. Communications of the ACM 27 (8). Pages 764-776. doi:10.1145/358198.358211 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1149. Ole Lehrmann Madsen, Boris Magnusson & Birger Mølier-Pedersen (1990): Strong typing of object-oriented languages revisited. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 140-150. doi:10.1145/97945.97964 Exclusion reasons: Q5 [III.ajk]This paper is analytical, with no aspiration to empiricity.

1150. Ole Lehrmann Madsen (2000): Towards a Unified Programming Language. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 1-26. doi:10.1007/3-540-45102-1_1 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1151. R. Maes (1978): On the representation of program structures by decision tables: a critical assessment. The Computer Journal 21 (4). Pages 290-295. doi:10.1093/comjnl/21.4.290 http://comjnl.oxfordjournals.org/content/21/4/290.abstract Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1152. Pattie Maes (1987): Concepts and experiments in computational reflection. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 147-155. doi:10.1145/38765.38821 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.

1153. Scott Malabarba, Raju Pandey, Jeff Gragg, Earl Barr & J. Fritz Barnes (2000): Runtime Support for Type-Safe Dynamic Java Classes. In Proc. ECOOP

2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 337-361. doi:10.1007/3-540-45102-1_17 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision – it is arguable whether the construction offered is a language design option at all, and even if it is, the performance evaluation is not of any relevance to our study.

1154. Jacques Malenfant, Guy Lapalme & Jean Vaucher (1989): ObjVProlog: Metaclasses in Logic. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 257-269. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890257.pdf Exclusion reasons: Q5 [III.ajk]This article introduces and illustrates a new construction. It has no empirical aspirations.

1155. J. Malenfant (1995): On the semantic diversity of delegation-based programming languages. In Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications. New York, NY, USA: ACM. OOPSLA '95. Pages 215-230. doi:10.1145/217838.217862 Exclusion reasons: Q5 [II.ajk]This article appears to be purely analytic, with no aspiration to empiricity.

1156. O. G. Mancino (1964): Characteristics of the FORTRAN CEP language. Communications of the ACM 7 (7). Pages 323-324. doi:10.1145/364520.364557 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1157. Linda Mannila & Michael de Raadt (2006): An objective comparison of languages for teaching introductory programming. In Proceedings of the 6th Baltic Sea conference on Computing education research: Koli Calling 2006. New York, NY, USA: ACM. Baltic Sea '06. Pages 32-37. doi:10.1145/1315803.1315811 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1158. Guillaume Marceau, Kathi Fisler & Shriram Krishnamurthi (2011): Do values grow on trees?: expression integrity in functional programming. In Proceedings of the seventh international workshop on Computing education research. New York, NY, USA: ACM. ICER '11. Pages 39-44. doi:10.1145/2016911.2016921 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1159. R. Marin, P. J. Sanz, P. Nebot & R. Wirz (2005): A multimodal interface to control a robot arm via the web: a case study on remote programming. Industrial Electronics, IEEE Transactions on 52 (6). Pages 1506-1520. doi:10.1109/TIE.2005.858733 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1160. Harry M. Markowitz, Ashok Malhotra & Donald P. Pazel (1984): The EAS-E application development system: principles and language summary. Communications of the ACM 27 (8). Pages 785-799. doi:10.1145/358198.358217 Exclusion reasons: Q5 [III.ajk]This exposition paper has no empirical ambitions.

1161. Shane Markstrum (2010): Staking claims: a history of programming language design claims and evidence: a positional work in progress. In Evaluation and Usability of Programming Languages and Tools. New York, NY, USA: ACM. PLATEAU '10. Pages 7:1-7:5. doi:10.1145/1937117.1937124 Exclusion reasons: Q1–2 [III.ajk]This article does not summarise or consolidate research on design decisions (it focuses on language-introducing papers).

1162. Shane Markstrum, Daniel Marino, Matthew Esquivel, Todd Millstein, Chris Andreae & James Noble (2010): JavaCOP: Declarative pluggable types for java. ACM Transactions on Programming Languages and Systems 32 (2). doi:10.1145/1667048.1667049 Exclusion reasons: Q1–2 [III.ajk]This article deals with a type system specification language, which is excluded as a PL under our protocol.

1163. Shane Markstrum, Emerson Murphy-Hill & Caitlin Sadowski (2012): Evaluation and usability of programming languages and tools (PLATEAU). In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. New York, NY, USA: ACM. Pages 219-220. doi:10.1145/2384716.2384778 Exclusion reasons: Q1–2 [III.ajk]This is not a study report.

1164. C. D. Marlin (1976): An experiment with the extensibility of SIMULA. SIGPLAN Notices 11. Pages 50-57. doi:10.1145/987335.987341 Exclusion reasons: Q1–2 [II.ajk]Language engineering study; no PL design issue under study for efficacy.

1165. Simon Marlow (2000): Writing High-Performance Server Applications in Haskell, Case Study: A Haskell Web Server. In Haskell Workshop. http://community.haskell.org/~simonmar/papers/web-server.ps.gz Exclusion reasons: Q5 [III.ajk]This article is an application demonstration. So far as it evaluates the language, it demonstrates an answer to a "can this be done" question and thus explores the implications of the design, making this a non-empirical work by our definition.

1166. Simon Marlow (2001): Developing High-Performance Server Applications in Haskell, Case Study: A Haskell Web Server. Electronic Notes in Theoretical Computer Science 41 (1). Pages 75-90. doi:10.1016/S1571-0661(05)80548-1 Exclusion reasons: Q1–2 [II.ajk]No comparison language.

1167. Lindsay Marshall & James Webber (2000): Gotos Considered Harmful and Other Programmers' Taboos. In PPIG 2000. (Found in http://ppig.org/workshops/12th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article is a discussion of programmer taboos and does not purport to report a study.

1168. Lindsay Marshall & Jim Webber (2002): The Misplaced Comma: Programmers' Tales and Traditions. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.

1169. Raul Marticorena, Carlos Lopez, Yania Crespo & F. Javier Perez (2010): Refactoring Generics in JAVA: A Case Study on Extract Method. In Software Maintenance and Reengineering, European Conference on. Los Alamitos, CA, USA: IEEE Computer Society. Pages 212-221. doi:10.1109/CSMR.2010.38 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1170. T. Martin (1981): Pearl at the age of five: Case study of development and application of a common high order realtime programming language. Computers in Industry 2 (1). Pages 1 - 11. doi:10.1016/0166-3615(81)90041-5 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1171. T. Martınez-Ruiz, F. Garcıa, M. Piattini & J. Münch (2011): Modelling software process variability: an empirical study. Software, IET 5 (2). Pages 172-187. doi:10.1049/iet-sen.2010.0020 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1172. Takeo Maruichi, Tetsuya Uchiki & Mario Tokoro (1987): Behavioral Simulation Based on Knowledge Objects. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 213-222. doi:10.1007/3-540-47891-4_20 Exclusion reasons: Q1–2 [II.ajk]It's not clear if there's a PL design question, but even if there is, there's no comparison.

1173. Fred A. Masterson (1985): Evaluating Logo: A Case Study in Requirements for Student Programming Languages. In Logo in the Schools.Hawort. Pages 179-195. Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.

1174. Hidehiko Masuhara & Akinori Yonezawa (1998): Design and partial evaluation of meta-objects for a concurrent reflective language. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 418-439. doi:10.1007/BFb0054102 Exclusion reasons: Q5 [III.ajk]This article's empirical content is restricted to implementation performance comparison.

1175. L. Shawn Matott, Kenny Leung & Junyoung Sim (2011): Application of MATLAB and Python optimizers to two case studies involving groundwater flow and contaminant transport modeling. Computers & Geosciences 37 (11). Pages 1894-1899. doi:10.1016/j.cageo.2011.03.017 Exclusion reasons: Q1–2 [III.ajk]This article does not appear to actually evaluate the language design decisions involved.

1176. Satoshi Matsuoka & Satoru Kawai (1988): Using tuple space communication in distributed object-oriented languages. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 276-284. doi:10.1145/62083.62108 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1177. Satoshi Matsuoka, Takuo Watanabe & Akinori Yonezawa (1991): Hybrid group reflective architecture for object-oriented concurrent reflective programming. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 231-250. doi:10.1007/BFb0057025 Exclusion reasons: Q5 [III.ajk]This article has no empirical aspiration.

1178. Satoshi Matsuoka, Kenjiro Taura & Akinori Yonezawa (1993): Highly efficient and encapsulated re-use of synchronization code in concurrent object-oriented languages. In OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. Pages 109-126. doi:10.1145/165854.165875 Exclusion reasons: Q3 [III.ajk]This article mostly does not aspire to empiricity. The benchmark is discussed summarily.

1179. Satoshi Matsuoka & Shigeo Itou (1998): Is Java Suitable for Portable High-Performance Computing? Preliminary Reports on Benchmarking Different Java Platforms. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 581. doi:10.1007/3-540-49255-0_149 Exclusion reasons: Q1–2 [III.ajk]This article summarises empirical research on the speed differences between different Java implementations. Although C implementations are used as well (with equivalent C programs), they serve as controls of implementation efficiency, and there is no PL design issue here.

1180. Jacob Matthews & Robert Bruce Findler (2009): perational semantics for multi-language programs. ACM Transactions on Programming Languages and Systems 31 (3). doi:10.1145/1498926.1498930 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1181. R. A. Maxion & R. T. Olszewski (2000): Eliminating exception handling errors with dependability cases: a comparative, empirical study. Software Engineering, IEEE Transactions on 26 (9). Pages 888-906. doi:10.1109/32.877848 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.

1182. Richard E. Mayer (1979): A psychology of learning BASIC. Communications of the ACM 22 (11). Pages 589-593. doi:10.1145/359168.359171 Exclusion reasons: Q1–2 [II.ajk]No relevance to programming language design.

1183. Richard E. Mayer & Piraye Bayman (1981): Psychology of calculator languages: a framework for describing differences in users' knowledge. Communications of the ACM 24 (8). Pages 511-520. doi:10.1145/358722.358735 Exclusion reasons: Q1–2 [II.ajk]No PL relevance.

1184. Richard E. Mayer, Jennifer L. Dyck & William Vilberg (1986): Learning to program and learning to think: what's the connection?. Communications of the ACM 29 (7). Pages 605-610. doi:10.1145/6138.6142 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1185. C. Mayer (2011): An empirical study of possible effects of static type systems on documentation – a controlled experiment with an undocumented application programming interface. . Bachelor's Thesis. Exclusion reasons: Q3 [III.ajk]Unpublished thesis.

1186. Jeff McAffer (1995): Meta-level Programming with CodA. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 190-214. doi:10.1007/3-540-49538-X_10 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1187. John McCarthy (1960): Recursive functions of symbolic expressions and their computation by machine, Part I. Communications of the ACM 3 (4). Pages 184-195. doi:10.1145/367177.367199 Exclusion reasons: Q5 [III.ajk]This language exposition has no aspiration to empiricity.

1188. JOHN C McCARTHY, ENDA FALLON & LIAM BANNON (2000): Dialogues on function allocation. International Journal of Human-Computer Studies 52 (2). Pages 191-201. doi:10.1006/ijhc.1999.0284 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate efficacy of anything.

1189. Daniel L. McCue (1992): Developing a class hierarchy for object-oriented transaction processing. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 413-426. doi:10.1007/BFb0053049 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1190. Sean McDirmid & Wilson C. Hsieh (2006): SuperGlue: Component Programming with Object-Oriented Signals. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 206-229. doi:10.1007/11785477_15 Exclusion reasons: Q5 [III.ajk]This article presents and analyzes a new language. It also presents a small "case study" which compares SuperGlue and Java by implementing the same program in both; such a study explores the implications of the design of the language and thus isn't empirical by our definition.

1191. Sean McDirmid (2007): Living it up with a live programming language. In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. Pages 623-638. doi:10.1145/1297027.1297073 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity (the Experience section is more analytic than empirical as it demonstrates what can be done, not what happens to be).

1192. James R. McGraw (1982): The VAL Language: Description and Analysis. ACM Transactions on Programming Languages and Systems 4 (1). Pages 44-82. doi:10.1145/357153.357157 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1193. M. Douglas McIlroy (1960): Macro instruction extensions of compiler languages. Communications of the ACM 3 (4). Pages 214-220. doi:10.1145/367177.367223 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricality.

1194. L. McIver & D. Conway (1996): Seven deadly sins of introductory programming language design. In Software Engineering: Education and Practice, 1996. Proceedings. International Conference. Pages 309-316. doi:10.1109/SEEP.1996.534015 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1195. Linda McIver (2002): Evaluating Languages and Environments for Novice Programmers. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This is a methodology paper, not a study.

1196. R. M. McKeag & P. Milligan (1980): An experiment in parallel program design. Software: Practice and Experience 10 (9). Pages 687-696. doi:10.1002/spe.4380100902 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1197. Ruth McKeever, Kevin McDaid & Brian Bishop (2009): Can Named Ranges Improve the Debugging Performance of Novice Spreadsheet Users?. In PPIG 2009. (Found in http://ppig.org/workshops/21st-programme.html.) Exclusion reasons: Q1–2 [III.ajk]Spreadsheets are not a textual language, hence exclude.

1198. Erik Meijer, Nigel Perry & Arjan van Yzendoorn (2001): Scripting .NET Using Mondrian. In Proc. ECOOP 2001 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2072. Pages 150-164. doi:10.1007/3-540-45337-7_9 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1199. Paola Mello & Antonio Natali (1987): Objects as Communicating Prolog Units. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 181-191. doi:10.1007/3-540-47891-4_17 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1200. Hayden Melton & Ewan Tempero (2007): An empirical study of cycles among classes in Java. Empirical Software Engineering 12 (4). Pages 389-415. doi:10.1007/s10664-006-9033-1 Exclusion reasons: Q1–2 [II.ajk]Studies a programmer pattern, no PL design relevance.

1201. H. Melton & E. Tempero (2007): Static Members and Cycles in Java Software. In First international symposium on Empirical Software Engineering and Measurement ESEM 2007. Pages 136-145. doi:10.1109/ESEM.2007.25 Exclusion reasons: Q1–2 [III.ajk]This article studies a program corpus to determine whether a particular programming pattern is associated with a particular quality-relevant attribute. It does not evaluate a language design decision.

1202. Anurag Mendhekar, Gregor Kiczales, John Lamping & John Lamping (1997): RG: A Case-Study for Aspect-Oriented Programming. SPL97-009 P9710044 at Xerox Palo Alto Research Center. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.8053 Exclusion reasons: Q5 [III.ajk]This article compares object-orientation and aspect-orientation in the context of CLOS by first writing a specific program in the OOp style, then identifying problems in it and the difficulty of solving them in the OO style, and finally solving them using AOP style. The use of a single program written for this study makes this less an empirical study and more the analytical exploration of the implications of the two styles, resulting in a result of possibility rather than a result of contingent truth. As such, this study is not empirical,

1203. Bertrand Meyer (1986): Genericity versus inheritance. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 391-405. doi:10.1145/28697.28738 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1204. B. Meyer (1987): Reusability: The Case for Object-Oriented Design. IEEE Software 4 (2). Pages 50-64. doi:10.1109/MS.1987.230097 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1205. Bertrand Meyer (2005): Attached Types and Their Application to Three Open Problems of Object-Oriented Programming. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 1-32. doi:10.1007/11531142_1 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1206. Mira Mezini (1997): Dynamic object evolution without name collisions. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 190-219. doi:10.1007/BFb0053380 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1207. Josephine Micallef & Gail E. Kaiser (1994): Extending attribute grammars to support programming-in-the-large. ACM Transactions on Programming Languages and Systems 16 (5). Pages 1572-1612. doi:10.1145/186025.186091 Exclusion reasons: Q1–2 [II.ajk]Development of attribute grammars, no evaluation of a PL design decision.

1208. Martin Mikelsons (1975): Computer assisted application definition. In Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 233-242. doi:10.1145/512976.512999 Exclusion reasons: Q1–2 [III.ajk]This article is a programming system exposition.

1209. J. R. Millenson (1970): Language and List Structure of a Compiler for Experimental Control. The Computer Journal 13 (4). Pages 340-343. doi:10.1093/comjnl/13.4.340 http://comjnl.oxfordjournals.org/content/13/4/340.abstract Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1210. Lance A. Miller (1974): Programming by non-programmers. International Journal of Man-Machine Studies 6 (2). Pages 237-260. doi:10.1016/S0020-7373(74)80004-0 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1211. Mark L. Miller (1979): A structured planning and debugging environment for elementary programming. International Journal of Man-Machine Studies 11 (1). Pages 79-95. doi:10.1016/S0020-7373(79)80006-1 Exclusion reasons: Q1–2 [II.ajk]This article deals with an interactive programming system, not a language as we have defined the concept.

1212. Robert Miller & Anand Tripathi (1997): Issues with exception handling in object-oriented systems. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 85-103. doi:10.1007/BFb0053375 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.

1213. Robert E. Millstein (1973): Control structures in Illiac IV Fortran. Communications of the ACM 16 (10). Pages 621-627. doi:10.1145/362375.362398 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.

1214. Todd Millstein & Craig Chambers (1999): Modular Statically Typed Multimethods. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 279-303. doi:10.1007/3-540-48743-3_13 Exclusion reasons: Q1–2 [II.ajk]Type-theoretical work.

1215. Todd Millstein, Mark Reay & Craig Chambers (2003): Relaxed MultiJava: balancing extensibility and modular typechecking. In OOPSLA '03: Proceedings of the 18th annual ACM SIGPLAN conference on Object-oriented programing, systems, languages, and applications. Pages 224-240. doi:10.1145/949305.949325 Exclusion reasons: Q5 [III.ajk]This article's empirical work is about implementation cost, not language design decisions.

1216. Robin Milner (2001): Computational flux. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 220-221. doi:10.1145/360204.360222 Exclusion reasons: Q5 [III.ajk]This lecture abstract does not report an empirical study.

1217. Walter Milner (2008): A Loop is a Compression. In PPIG 2008. (Found in http://ppig.org/workshops/20th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.

1218. Naftaly H. Minsky (1984): Selective and locally controlled transport of privileges. ACM Transactions on Programming Languages and Systems 6 (4).

Pages 573-602. doi:10.1145/1780.1786 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper has no empirical content.

1219. Naftaly H. Minsky (1996): Towards alias-free pointers. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 189-209. doi:10.1007/BFb0053062 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

1220. Yaron M. Minsky (2008): Caml trading. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 285. doi:10.1145/1328438.1328441 Exclusion reasons: Q3 Q5 [III.ajk]This is an abstract of a talk that reported industrial experience.

1221. Rajiv Mirani & Paul Hudak (2004): First-class monadic schedules. ACM Transactions on Programming Languages and Systems 26 (4). Pages 609-651. doi:10.1145/1011508.1011509 Exclusion reasons: Q5 [III.ajk]This article's only empirical aspiration relates to implementation performance.

1222. Salman Mirghasemi, John J. Barton & Claude Petitpierre (2011): Naming anonymous javascript functions. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 277-288. doi:10.1145/2048147.2048222 Exclusion reasons: Q1–2 [II.ajk]This article deals with a debugging aid.

1223. Jayadev Misra (1994): Powerlist: a structure for parallel recursion. ACM Transactions on Programming Languages and Systems 16 (6). Pages 1737-1767. doi:10.1145/197320.197356 Exclusion reasons: Q1–2 [II.ajk]Implementation technique.

1224. J. C. Mitchell & R. Harper (1988): The essence of ML. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 28-46. doi:10.1145/73560.73563 Exclusion reasons: Q5 [II.ajk]Theoretical work.

1225. Jeffrey Mitchell & Charles Welty (1988): Experimentation in computer science: an empirical view. International Journal of Man-Machine Studies 29 (6). Pages 613-624. doi:10.1016/S0020-7373(88)80069-5 Exclusion reasons: Q1–2 [II.ajk]This article does not seem to concern itself with language design decision evaluation.

1226. John Mitchell, Sigurd Meldal & Neel Madhav (1991): An extension of standard ML modules with subtyping and inheritance. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 270-278. doi:10.1145/99583.99620 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1227. Francesmary Modugno, T. R. G. Green & Brad A. Myers (1994): Visual programming in a visual domain: a case study of cognitive dimensions. In Proceedings of the conference on People and computers IX. New York, NY, USA: Cambridge University Press. Pages 91-108. http://www.cs.cmu.edu/~garnet/pbd-group/papers/hci94.pdf Exclusion reasons: Q1–2 [II.ajk]Visual languages are not programming languages by our definition.

1228. Thomas Moher & G. Michael Schneider (1982): Methodology and experimental research in software engineering. International Journal of Man-Machine Studies 16 (1). Pages 65-87. doi:10.1016/S0020-7373(82)80072-2 Exclusion reasons: Q1–2 [III.ajk]This article discusses and review methodological issues in programmer studies; it does not summarise or consolidate the actual primary studies.

1229. A. Molesini, A. Garcia, C.F.G. von Chavez & T. Batista (2008): On the Quantitative Analysis of Architecture Stability in Aspectual Decompositions. In Software Architecture, 2008. WICSA 2008. Seventh Working IEEE/IFIP Conference on. Pages 29-38. doi:10.1109/WICSA.2008.26 Exclusion reasons: Q1–2 [III.ajk]This article deals with architecture, not implementation language issues.

1230. Christopher Monsanto, Nate Foster, Rob Harrison & David Walker (2012): A compiler and run-time system for network programming languages. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 217-230. doi:10.1145/2103656.2103685 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate efficacy.

1231. C. Montangero, G. Pacini & F. Turini (1977): Two-level control structure for nondeterministic programming. Communications of the ACM 20 (10). Pages 725-730. doi:10.1145/359842.359850 Exclusion reasons: Q5 [III.ajk]This article is constructive-analytical; there is no attempt at empiricity.

1232. Miguel P. Monteiro (2011): On the cognitive foundations of modularity. In PPIG 2011. http://ppig.org/papers/23/32%20Monteiro.pdf Exclusion reasons: Q5 [II.ajk]Does not seem to do any empirical work.

1233. G. Monteleone (1989): Generalized conjunctive types. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 242-249. doi:10.1145/75277.75298 Exclusion reasons: Q5 [II.ajk]Formal type theoretical development only.

1234. Calvin N. Mooers (1966): TRAC, a procedure-describing language for the reactive typewriter. Communications of the ACM 9 (3). Pages 215-219. doi:10.1145/365230.365270 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1235. Calvin N. Mooers (1968): Standards: Accommodating standards and identification of programming languages. Communications of the ACM 11 (8). Pages 574-576. doi:10.1145/363567.364061 Exclusion reasons: Q5 [III.ajk]This relatively short note does not aspire to empiricity.

1236. David A. Moon (1986): Object-oriented programming with flavors. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 1-8. doi:10.1145/28697.28698 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1237. Adriaan Moors, Frank Piessens & Martin Odersky (2008): Generics of a higher kind. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 423-438. doi:10.1145/1449764.1449798 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.

1238. Floréal Morandat, Brandon Hill, Leo Osvald & Jan Vitek (2012): Evaluating the Design of the R Language. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 104-131. doi:10.1007/978-3-642-31057-7_6 Exclusion reasons: Q1–2 [III.ajk]This article is a language critique that does not discuss design decision efficacy.

1239. Luc Moreau & Christian Queinnec (2005): Resource aware programming. ACM Transactions on Programming Languages and Systems 27 (3). Pages 441-476. doi:10.1145/1065887.1065891 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1240. M. Moreaux & R. Y. Lorin (1991): Communication between heterogeneous machines: A case study, implementation in Ada. In Industrial Electronics, Control and Instrumentation, 1991. Proceedings. IECON '91., 1991 International Conference on. Pages 928-931 vol.2. doi:10.1109/IECON.1991.239166 Exclusion reasons: Q5 [III.ajk]This article does not seem to aspire to empiricity.

1241. J. E. Moreira & S. P. Midkiff (1998): Fortran 90 in CSE: a case study. Computational Science Engineering, IEEE 5 (2). Pages 39-49. doi:10.1109/99.683741 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which a particular computational problem was coded in both Fortran 90 and C++, and the benefits of each language was assessed by analytical comparison and performance measurements. While it may be possible to view this study as evaluating the design decision between certain C++ and certain Fortran 90 features, this is not its stated intention and as the study is not properly focused in that way, it is useless from that perspective.

1242. J.E. Moreira & S.P. Midkiff (1998): A Case Study of Fortran in Computational Science and Engineering. IBM Research Report. http://domino.watson.ibm.com/library/cyberdig.nsf/a3807c5b4823c53f85256561006324be/c4611566c95fe8cc852565b00060d276?OpenDocument Exclusion reasons: Q5 [III.ajk]This article reports a study in which a program was written in C++ and Fortran, and their performance was compared. Arguably, this could be taken as an empirical evaluation of the efficacy of the differences between the languages, but the study clearly is focused only on an analytical comparison of the language design decisions, and empirical comparison of implementations. The empirical efficacy evaluation is too underreported to be counted.

1243. J. Moreira, S. Midkiff, M. Gupta & R. Lawrence (1999): High Performance Computing with the Array Package for Java: A Case Study using Data Mining. In Supercomputing, ACM/IEEE 1999 Conference. Pages 10. doi:10.1109/SC.1999.10025 Exclusion reasons: Q1–2 [III.ajk]This article compares empirically the performance of several different implementations of the same basic algorithm, in both Java and Fortran. There appears to be no evaluation of language designs.

1244. Carroll Morgan (1988): The specification statement. ACM Transactions on Programming Languages and Systems 10 (3). Pages 403-419. doi:10.1145/44501.44503 Exclusion reasons: Q5 [III.ajk]This analytic-constructive article does not aspire to empiricity.

1245. James H. Morris, Jr. (1973): Protection in programming languages. Communications of the ACM 16 (1). Pages 15-21. doi:10.1145/361932.361937 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1246. James H. Morris, Eric Schmidt & Philip Wadler (1980): Experience with an applicative string processing language. In Proc. 7th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 32-46. doi:10.1145/567446.567450 Exclusion reasons: Q5 [III.ajk]This analytical evaluative paper does not aspire to empiricity.

1247. R. Morrison, A. Dearle, R. C. H. Connor & A. L. Brown (1991): An ad hoc approach to the implementation of polymorphism. ACM Transactions on Programming Languages and Systems 13 (3). Pages 342-371. doi:10.1145/117009.117017 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1248. Angelo Morzenti & Pierluigi San Pietro (1988): An object-oriented logic language for modular system specification. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 39-58. doi:10.1007/BFb0057014 Exclusion reasons: Q1–2 [II.ajk]Studies a specification language, not a programming language.

1249. S. Moser & O. Nierstrasz (1996): The effect of object-oriented frameworks on developer productivity . Computer 29 (9). Pages 45-51. doi:10.1109/2.536783 Exclusion reasons: Q1–2 [II.ajk]This article studies metrics and does not evaluate any language design decisions.

1250. J. Eliot B. Moss & Walter H. Kohler (1987): Concurrency Features for the Trellis/Owl Language. In Proc. ECOOP'87 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 276. Pages 171-180. doi:10.1007/3-540-47891-4_16 Exclusion reasons: Q5 [III.ajk]This article does not empirically evaluate its construct for efficacy.

1251. Ana Lúcia De Moura & Roberto Ierusalimschy (2009): Revisiting coroutines. ACM Transactions on Programming Languages and Systems 31 (2). doi:10.1145/1462166.1462167 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1252. Leonard J. Mselle (2012): Learning Programming by using Memory Transfer Language (MTL) Without the Intervention of an Instructor . In PPIG 2012. Exclusion reasons: Q1–2 [III.ajk]This article deals with a visualization tool, not a language design issue.

1253. T.R. Muck, M. Gernoth, W. Schroder-Preikschat & A.A. Frohlich (2011): A Case Study of AOP and OOP Applied to Digital Hardware Design. In Computing System Engineering (SBESC), 2011 Brazilian Symposium on. Pages 66-71. doi:10.1109/SBESC.2011.23 Exclusion reasons: Q1–2 [III.ajk]HDLs are not programming languages.

1254. Warwick B. Mugridge, John Hamer & John G. Hosking (1991): Multi-methods in a statically-typed programming language. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 307-324. doi:10.1007/BFb0057029 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

1255. Paul Mulholland (1995): Prolog without tears: an evaluation of the effectiveness of a non Byrd Box model for students. In PPIG 1995. (Found in http://ppig.org/workshops/7th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]Full text copy retrieved from http://people.kmi.open.ac.uk/paulm/sv-papers/ppig95.ps. This article evaluates a system that shows students how a Prolog program executes. It does not evaluate a language design decision.

1256. P. Mulholland (1998): A Principled Approach to the Evaluation of Software Visualization: a case-study in Prolog. In M. Brown and J. Dominique and B. Price and J. Stasko (ed.) Software Visualization: Programming as a multi-media experience. Cambridge, MA: MIT Press. Exclusion reasons: Q1–2 [II.ajk]Based on the title, no PL design issue.

1257. H. Muller, J. Rose, J. Kempf & T. Stansbury (1989): The use of multimethods and method combination in a CLOS based window interface. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 239-253. doi:10.1145/74877.74903 Exclusion reasons: Q1–2 [II.ajk]No language comparison.

1258. Robert Muller (1992): M-LISP: a representation-independent dialect of LISP with reduction semantics. ACM Transactions on Programming Languages and Systems 14 (4). Pages 589-616. doi:10.1145/133233.133254 Exclusion reasons: Q1–2 [II.ajk]Theoretical development.

1259. Stefan Muller & Stephen Chong (2012): Towards a practical secure concurrent language. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 57-74. doi:10.1145/2384616.2384621 Exclusion reasons: Q5 [II.ajk]The abstract does not reveal any empirical evaluation work.

1260. Stephan Murer, Stephen Omohundro, David Stoutamire & Clemens Szyperski (1996): Iteration abstraction in Sather. ACM Transactions on Programming Languages and Systems 18 (1). Pages 1-15. doi:10.1145/225540.225541 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity (the experience reports included may be empirical but they do not address decision efficacy because there's no comparison).

1261. Susan C. Murphy, Per Gunningberg & John P. J. Kelly (1991): Experiences with Estelle, LOTOS and SDL: a protocol implementation experiment. Computer Networks and ISDN Systems 22 (1). Pages 51-59. doi:10.1016/0169-7552(91)90081-M Exclusion reasons: Q1–2 [II.ajk]Study of specification; so far as there is any PL design issue, there is no comparison.

1262. Emerson R. Murphy-Hill & Andrew P. Black (2004): Traits: experience with a language feature. In OOPSLA '04: Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. Pages 275-282. doi:10.1145/1028664.1028771 Exclusion reasons: Q5 [III.ajk]This article evaluates traits analytically; there is no real empirical content.

1263. Emerson R. Murphy-Hill, Philip J. Quitslund & Andrew P. Black (2005): Removing duplication from java.io: a case study using traits. In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 282-291. doi:10.1145/1094855.1094963 Exclusion reasons: Q5 [III.ajk]This article reports an analytical evaluation of traits.

1264. Chet Murthy (2007): Advanced programming language design in enterprise software: a lambda-calculus theorist wanders into a datacenter. In Proc. 34th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 263-264. doi:10.1145/1190216.1190255 Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.

1265. Radu Muschevici, Alex Potanin, Ewan Tempero & James Noble (2008): Multiple dispatch in practice. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 563-582. doi:10.1145/1449764.1449808 Exclusion reasons: Q1–2 [III.ajk]This empirical paper describes the actual use of multiple dispatch in several languages providing support for it and the use of its emulation patterns in Java. This is a purely descriptive study, and does not evaluate anything.

1266. Eugene W. Myers (1984): Efficient applicative data types. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 66-75. doi:10.1145/800017.800517 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1267. Brad A. Myers (1990): Creating user interfaces using programming by example, visual programming, and constraints. ACM Transactions on Programming Languages and Systems 12 (2). Pages 143-177. doi:10.1145/78942.78943 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1268. Andrew C. Myers (1999): JFlow: practical mostly-static information flow control. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 228-241. doi:10.1145/292540.292561 Exclusion reasons: Q5 [III.ajk]This article has no empirical aspirations.

1269. Clayton Myers & Elisa Baniassad (2009): Silhouette: visual language for meaningful shape. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 917-924. doi:10.1145/1639950.1640057 Exclusion reasons: Q1–2 [II.ajk]Visual languages aren't by our definition.

1270. Barbee T. Mynatt (1984): The effect of semantic complexity on the comprehension of program modules. International Journal of Man-Machine Studies 21 (2). Pages 91-103. doi:10.1016/S0020-7373(84)80060-7 Exclusion reasons: Q1–2 [III.ajk]This article evaluates comprehension and semantic complexity of programs; it does not have a clear language design issue at stake.

1271. Mika Mäntylä (2003): Bad smells in software – a taxonomy and an empirical study. . Master's Thesis at Helsinki University of Technology. http://www.soberit.hut.fi/sems/shared/deliverables_public/mmantyla_thesis_final.pdf Exclusion reasons: Q1–2 [II.ajk]This study deals with bad code smell, not any language design issues.

1272. Gerhard Müller & Anna-Kristin Pröfrock (1989): Four Steps and a Rest in Putting an Object-Oriented Programming Environment to Practical Use. In Proc. ECOOP'89 European Conference on Object-Oriented Programming.Cambridge University Press. Pages 271-282. http://www.ifs.uni-linz.ac.at/~ecoop/cd/papers/ec89/ec890271.pdf Exclusion reasons: Q5 [III.ajk]This article does not evaluate any language design decisions.

1273. Karl Naden, Robert Bocchino, Jonathan Aldrich & Kevin Bierhoff (2012): A type system for borrowing permissions. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 557-570. doi:10.1145/2103656.2103722 Exclusion reasons: Q5 [II.ajk]No empirical evaluation (example illustrations are very likely analytical).

1274. G. Nagy & M. Carlson Pennebaker (1974): A step toward automatic analysis of student programming errors in a batch environment. International Journal of Man-Machine Studies 6 (5). Pages 563-578. doi:10.1016/S0020-7373(74)80018-0 Exclusion reasons: Q1–2 [II.ajk]This article deals only with methodological tool support.

1275. Sebastian Nanz, Faraz Torshizi, Michela Pedroni & Bertrand Meyer (2013): Design of an empirical study for comparing the usability of concurrent programming languages. Information and Software Technology 55 (7). doi:10.1016/j.infsof.2012.08.013 Exclusion reasons: Q3 [III.ajk]There is no doubt about inclusion here. [posthoc]Published in 2013 and thus out of our range. Originally included as an in-press publication dated in 2012.

1276. H. Albert Napier, Richard R. Batsell, Norman S. Guadango & David M. Lane (1989): Impact of a restricted natural language interface on ease of learning and productivity. Communications of the ACM 32 (10). Pages 1190-1198. doi:10.1145/67933.67936 Exclusion reasons: Q1–2 [III.ajk]This article reports a study comparing two different interactive user interfaces. Neither is a programming language by our definition nor does the study seem to be transferable to a PL design context.

1277. Pedro Hugo do Nascimento Gabriel (2010): Software languages engineering: experimental evaluation. . MSc at Universidade Nova de Lisboa Faculdade de Ciências e Tecnologia Departamento de Informática. http://run.unl.pt/handle/10362/4854 Exclusion reasons: Q1–2 [II.ajk]Studies language evaluation at a meta-level.

1278. Emal Nasseri (2009): An empirical investigation of inheritance trends in JAVA OSS evolution. . PhD at Brunel University. http://bura.brunel.ac.uk/handle/2438/3643 Exclusion reasons: Q1–2 [II.ajk]Program evolution, not language design.

1279. E. Nasseri, S. Counsell & M. Shepperd (2010): Class movement and re-location: An empirical study of Java inheritance evolution. Journal of Systems and Software 83 (2). Pages 303-315. doi:10.1016/j.jss.2009.08.011 Exclusion reasons: Q1–2 [II.ajk]Evaluation of program evolution.

1280. Peter Naur, J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden & M. Woodger (1960): Report on the algorithmic language ALGOL 60. Communications of the ACM 3 (5). Pages 299-314. doi:10.1145/367236.367262 Exclusion reasons: Q1–2 [III.ajk]This language specification does not report an evaluative study.

1281. RAQUEL NAVARRO-PRIETO & JOSE J. CAÑAS (2001): Are visual programming languages better? The role of imagery in program comprehension. International Journal of Human-Computer Studies 54 (6). Pages 799-829. doi:10.1006/ijhc.2000.0465 Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded.

1282. Farshad Nayeri, Ben Hurwitz & Frank Manola (1994): Generalizing dispatching in a distributed object system. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 450-473. doi:10.1007/BFb0052196 Exclusion reasons: Q5 [III.ajk]This article includes an analysis (styled experiment) of the language in question and does not aspire to empiricity.

1283. T. Ndoa & C. Jia (2011): Empirical Case Study of Measuring Productivity of Programming Language Ruby and Ruby on Rails. In ICSEA 2011, The Sixth International Conference on Software Engineering Advances. Pages 367-368. http://www.thinkmind.org/index.php?view=article&articleid=icsea_2011_15_30_10268 Exclusion reasons: Q3 [III.ajk]This article is too short on detail, and I was unable to find a better one.

1284. George C. Necula, Scott McPeak & Westley Weimer (2002): CCured: type-safe retrofitting of legacy code. In Proc. 29th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 128-139. doi:10.1145/503272.503286 Exclusion reasons: Q1–2 [III.ajk]This article includes an evaluation section where existing C programs are presented (unchanged) to CCured. This does not evaluate efficacy as we have designed it.

1285. Matthias Neubauer, Peter Thiemann, Martin Gasbichler & Michael Sperber (2002): Functional logic overloading. In Proc. 29th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 233-244. doi:10.1145/503272.503294 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1286. Gustaf Neumann & Uwe Zdun (2002): Pattern-Based Design and Implementation of an XML and RDF Parser and Interpreter: A Case Study. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 392-414. doi:10.1007/3-540-47993-7_17 Exclusion reasons: Q1–2 [III.ajk]This design presentation does not report an evaluative study.

1287. Christian Neusius (1991): Synchronizing Actions. In Proc. ECOOP'91 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 512. Pages 118-132. doi:10.1007/BFb0057018 Exclusion reasons: Q1–2 [II.ajk]Language feature "framework" development only.

1288. Seppo Nevalainen & Jorma Sajaniemi (2005): Short-Term Effects of Graphical versus Textual Visualisation of Variables on Program Perception. In PPIG 2005. (Found in http://ppig.org/workshops/17th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article reports a human-subject experiment to evaluate a particular program-visualization tool. Since in this study non-textual languages are excluded, this study is out of scope.

1289. A. Newell & F. M. Tonge (1960): An introduction to information processing language V. Communications of the ACM 3 (4). Pages 205-211. doi:10.1145/367177.367205 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1290. William M. Newman (1970): An experimental display programming language for the PDP-10 computer. at UTAH UNIV SALT LAKE CITY COMPUTER SCIENCE DIV. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=AD0762010 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1291. Julian Newman (1977): The processing of two types of command statement: A contribution to cognitive ergonomics.. IEEE Transactions on Systems, Man, & Cybernetics. doi:10.1109/TSMC.1977.4309645 Exclusion reasons: Q1–2 [III.ajk]This article does not study any language design decisions.

1292. Khuong A. Nguyen (2011): A case study on the usability of NXT-G programming language. In PPIG 2011. http://ppig.org/papers/23/22%20Nguyen.pdf Exclusion reasons: Q1–2 [III.ajk]This article evaluates a visual programming language, which is excluded.

1293. Yang Ni, Adam Welc, Ali-Reza Adl-Tabatabai, Moshe Bach, Sion Berkowits, James Cownie, Robert Geva, Sergey Kozhukow, Ravi Narayanaswamy, Jeffrey Olivier, Serguei Preis, Bratin Saha, Ady Tal & Xinmin Tian (2008): Design and implementation of transactional constructs for C/C++. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 195-212. doi:10.1145/1449764.1449780 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1294. Keiron Nicholson, Judith Good & Katy Howland (2009): Concrete Thoughts on Abstraction. In PPIG 2009. (Found in http://ppig.org/workshops/21st-programme.html.) Exclusion reasons: Q5 [III.ajk]This article does not evaluate language design decisions.

1295. Kjell W. Nielsen & Ken Shumate (1987): Designing large real-time systems with Ada. Communications of the ACM 30 (8). Pages 695-715. doi:10.1145/27651.27655 Exclusion reasons: Q5 [III.ajk]This article includes a "case study" which is properly described as an analytical example.

1296. Hanne Riis Nielson & Flemming Nielson (1994): Higher-order concurrent programs with finite communication topology (extended abstract). In Proc. 21th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 84-97. doi:10.1145/174675.174538 Exclusion reasons: Q1–2 [II.ajk]Program analysis.

1297. O. M. Nierstrasz (1987): Active objects in hybrid. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 243-253. doi:10.1145/38765.38829 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1298. Oscar Nierstrasz & Michael Papathomas (1990): Viewing object as patterns of communicating agents. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 38-43. doi:10.1145/97945.97952 Exclusion reasons: Q1–2 [II.ajk]Development of a language development approach.

1299. Rob V. Van Nieuwpoort, Gosia Wrzesińska, Ceriel J. H. Jacobs & Henri E. Bal (2010): Satin: A high-level and efficient grid programming model. ACM Transactions on Programming Languages and Systems 32 (3). doi:10.1145/1709093.1709096 Exclusion reasons: Q1–2 [III.ajk]It is debatable whether the construction discussed is a (set of) language design decision(s). Even if it is, the evaluation is not comparative and thus must be excluded under our protocol.

1300. Uolevi Nikula, Jorma Sajaniemi, Matti Tedre & Stuart Wray (2007): Python and roles of variables in introductory programming: experiences from three educational institutions. Journal of Information Technology Education 6. Pages 199-214 . http://www.jite.org/documents/Vol6/JITEv6p199-214Nikula269.pdf Exclusion reasons: Q1–2 [II.ajk]This article does not seem to evaluate any language design decisions.

1301. K. Nishimura (2009): Empirical evaluation of object-oriented programming effectiveness in different types of program. In ICCAS-SICE, 2009. Pages 5537-5543. Exclusion reasons: Q1–2 [III.ajk]This rather confusing paper appears to be an analytical comparison of OO to procedural programming. No actual language design decisions appear to be involved.

1302. Hiroki Nishino (2011): Misfits in abstractions: towards user-centered design in domain-specific languages for end-user programming. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 215-216. doi:10.1145/2048147.2048214 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1303. Ronald J. Norman & Jr. Jay F. Nunamaker (1989): CASE productivity perceptions of software engineering professionals. Communications of the ACM 32 (9). Pages 1102-1108. doi:10.1145/66451.66458 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1304. Charles D. Norton, Viktor Decyk & Joan Slottow (1998): Applying Fortran 90 and Object-Oriented Techniques to Scientific Applications. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 581. doi:10.1007/3-540-49255-0_150 Exclusion reasons: Q7 [III.ajk]A longer version of this paper is available at http://hdl.handle.net/2014/19305. Ref 7 in the longer version is potentially empirical and its results are briefly discussed. [posthoc] Initially included as study S109, but EXCLUDED post hoc. Close reading reveals no discussion of empirical evidence in the primary studies.

1305. Gary J. Nutt (1978): A comparison of PASCAL and FORTRAN as introductory programming languages. SIGPLAN Notices 13 (2). Pages 57-62. doi:10.1145/953422.953425 Exclusion reasons: Q1–2 [II.ajk]Informal experience report by the author is to be excluded.

1306. Pamela O'Shea & Chris Exton (2003): Does the empirical evidence support visualisation?. In PPIG 2003. (Found in http://ppig.org/workshops/15th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article surveys studies on program comprehension. It does not survey language design issue evaluations.

1307. F.S. Ocariza, K. Pattabiraman & B. Zorn (2011): JavaScript Errors in the Wild: An Empirical Study. In Software Reliability Engineering (ISSRE), 2011 IEEE 22nd International Symposium on. Pages 100-109. doi:10.1109/ISSRE.2011.28 Exclusion reasons: Q1–2 [III.ajk]This article describes empirically errors in actual Javascript programs. There was no language design issue at stake.

1308. F.S. Ocariza (2012): Characterizing the JavaScript errors that occur in production web applications: an empirical study. University of British Columbia. https://circle.ubc.ca/handle/2429/42103 Exclusion reasons: Q1–2 [III.ajk]This master's thesis reports a software corpus study cataloguing programming errors. There is no efficacy question in play.

1309. Martin Odersky (1991): How to make destructive updates less destructive. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 25-36. doi:10.1145/99583.99590 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empricity.

1310. Martin Odersky & Konstantin Läufer (1996): Putting type annotations to work. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 54-67. doi:10.1145/237721.237729 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1311. Martin Odersky & Philip Wadler (1997): Pizza into Java: translating theory into practice. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 146-159. doi:10.1145/263699.263715 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1312. Martin Odersky (2004): The Scala Experiment – Can We Provide Better Language Support for Component Systems?. Volume 3302.In Chin, Wei-Ngan (ed.) Programming Languages and Systems. Lecture Notes in Computer Science. Pages 364-365. doi:10.1007/978-3-540-30477-7_24 Exclusion reasons: Q1–2 [III.ajk]This abstract does not report an evaluative study.

1313. Martin Odersky & Matthias Zenger (2005): Scalable component abstractions. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 41-57. doi:10.1145/1094811.1094815 Exclusion reasons: Q5 [III.ajk]The "case studies" answer questions of the form "can ... be done?" and as such explore the implications of established facts and thus are not empirical.

1314. Martin Odersky (2006): The Scala experiment: can we provide better language support for component systems?. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 166-167. doi:10.1145/1111037.1111052 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1315. William C Ogden (1997): Using natural language interfaces. In Handbook of human-computer interaction.Elsevier Science BV. Pages 137-161. Exclusion reasons: Q1–2 [II.ajk]This article reviews empirical research on natural-language user interfaces and does not concern itself with programming language design issues.

1316. Atsushi Ohori & Kazuhiko Kato (1993): Semantics for communication primitives in a polymorphic language. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 99-112. doi:10.1145/158511.158529 Exclusion reasons: Q1–2 [II.ajk]Theoretical language development.

1317. Hideaki Okamura & Yutaka Ishikawa (1994): Object location control using meta-level programming. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 299-319. doi:10.1007/BFb0052189 Exclusion reasons: Q5 [III.ajk]This article includes an arguably empirical performance evaluation. It, however, does not evaluate the language design decisions empirically.

1318. J.V. Oldfield (1986): Logic programs and an experimental architecture for their execution. Computers and Digital Techniques, IEE Proceedings E 133 (3). Pages 163-167. doi:10.1049/ip-e:19860021 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1319. Bruno C.d.S. Oliveira, Meng Wang & Jeremy Gibbons (2008): The visitor pattern as a reusable, generic, type-safe component. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 439-456. doi: 10.1145/1449764.1449799 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1320. José de Oliveira Guimarães (1998): Reflection for statically typed languages. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 440-461. doi:10.1007/BFb0054103 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1321. Walter G. Olthoff (1986): Augmentation of object-oriented programming by concepts of abstract data type theory: the ModPascal experience. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 429-443. doi:10.1145/28697.28742 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1322. Paul W. Oman & Curtis R. Cook (1990): Typographic style is more than cosmetic. Communications of the ACM 33 (5). Pages 506-520. doi:10.1145/78607.78611 Exclusion reasons: Q1–2 [III.ajk]This article reports a study evaluating program typography. There is no language design decision at play.

1323. Andrea Omicini & Antonio Natali (1994): Object-oriented computations in logic programming. In Proc. ECOOP'94 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 821. Pages 194-212. doi:10.1007/BFb0052184 Exclusion reasons: Q1–2 [II.ajk]Theory development, no comparison (based on the abstract).

1324. Ascher Opler (1965): Procedure-oriented language statements to facilitate parallel processing. Communications of the ACM 8 (5). Pages 306-307. doi:10.1145/364914.364947 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1325. Ascher Opler (1966): Requirements for real-time languages. Communications of the ACM 9 (3). Pages 196-199. doi:10.1145/365230.365265 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.

1326. Leif Oppermann (2008): On the choice of programming languages for developing location-based mobile games. Volume P-133.In INFORMATIK 2008 Beherrschbare Systeme -dank Informatik Band 1 P-133, 481-488 . Lecture Notes in Informatics. http://subs.emis.de/LNI/Proceedings/Proceedings133/article4500.html Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions

1327. Rachel Or-Bach & Ilana Lavy (2004): Cognitive activities of abstraction in object orientation: an empirical study. SIGCSE Bulletin 36 (2). Pages 82-86. doi:10.1145/1024338.1024378 Exclusion reasons: Q1–2 [III.ajk]This article has no relevance to programming language design.

1328. Francisco Ortin, Sheila Mendez, Vicente García-Díaz & Miguel Garcia (2012): On the suitability of dynamic languages for hot-reprogramming a robotics framework: a Python case study. Software: Practice and Experience. doi:10.1002/spe.2162 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1329. G. Ospina, F. Gobert & B. Le Charlier (2005): An Experiment in Programming Language Interoperability: Using a C Polyhedra Library with a Java Application. Universit{\'e} catholique de Louvain. D{\'e}partement d'Ing{\'e}nierie Informatique. (no URL known at this time.) Exclusion reasons: Q1–2 [III.ajk]Copy found at http://subversion.assembla.com/svn/pmbspace/pr5/trunk/papers/experi_c_j.ps – this paper does not report a comparative evaluation of language design decisions.

1330. Harold L. Ossher (1984): Grids: A new program structuring mechanism based on layered graphs. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 11-22. doi:10.1145/800017.800512 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1331. Klaus Ostermann & Mira Mezini (2001): Object-oriented composition untangled. In OOPSLA '01: Proceedings of the 16th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 283-299. doi:10.1145/504282.504303 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1332. Klaus Ostermann (2002): Dynamically Composable Collaborations with Delegation Layers. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 89-110. doi:10.1007/3-540-47993-7_4 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1333. Klaus Ostermann, Mira Mezini & Christoph Bockisch (2005): Expressive Pointcuts for Increased Modularity. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 214-240. doi:10.1007/11531142_10 Exclusion reasons: Q5 [II.ajk]Formal analysis.

1334. Krzysztof Ostrowski, Ken Birman, Danny Dolev & Jong Hoon Ahnn (2008): Programming with Live Distributed Objects. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 463-489. doi:10.1007/978-3-540-70592-5_20 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1335. Krzysztof Ostrowski, Chuck Sakoda & Ken Birman (2010): Self-Replicating Objects for Multicore Platforms. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 452-477. doi:10.1007/978-3-642-14107-2_22 Exclusion reasons: Q1–2 [III.ajk]There may be an incidental PL design issue here, but it is clear that the study is not focused on it. Instead, it presents and evaluates a concurrency model and its implementation.

1336. C Michael Overstreet & Richard E. Nance (1985): A specification language to assist in analysis of discrete event simulation models. Communications of the ACM 28 (2). Pages 190-201. doi:10.1145/2786.2792 Exclusion reasons: Q1–2 [II.ajk]Specification, not programming.

1337. F. Pachet & P. Roy (1995): Mixing Constraints and Objects: a Case Study in Automatic Harmonization. In TOOLS Europe '95.Prentice-Hall. Pages 119-126. http://www-poleia.lip6.fr/~fdp/MyPapers/BackTalk/contraintes-tools95.ps.Z Exclusion reasons: Q5 [III.ajk]This article does not, despite its title, aspire to empiricity.

1338. Frank Padberg (2000): Estimating the Impact of the Programming Language on the Development Time of a Software Project. In International Software Development and Management Conference AP-SEPG/ISDM. Pages 287-298. http://www.ipd.kit.edu/Tichy/publications.php?id=42 Exclusion reasons: Q1–2 [III.ajk]This article reports a statistical method for empirically estimating certain efficacy parameters of various languages. Although the paper includes a sample study, it is not its main contribution. Combining that with the fact that there is no clear language design decision at play (the comparison is of multiple languages together), this article cannot be said to evaluate the efficacy of any language design decision.

1339. Andreas Paepcke (1988): PCLOS: A Flexible Implementation of CLOS Persistence. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 374-389. doi:10.1007/3-540-45910-3_22 Exclusion reasons: Q1–2 [III.ajk]This article is a system exposition, with no attempt at evaluation.

1340. A. Paepcke (1989): PCLOS: a critical review. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 221-253. doi:10.1145/74877.74902 Exclusion reasons: Q5 [III.ajk]This analytical language review does not aspire to empiricity.

1341. Andreas Paepcke (1990): PCLOS: stress testing CLOS experiencing the metaobject protocol. In OOPSLA/ECOOP '90: Proceedings of the European conference on object-oriented programming and Object-oriented programming systems, languages, and applications. Pages 194-211. doi:10.1145/97945.97969 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1342. T. W. Page, Jr., S. Berson, W. Cheng & R. R. Muntz (1989): An object-oriented modelling enviornment. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 287-296. doi:10.1145/74877.74907 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1343. Christophe Pallier, Emmanuel Dupoux & Xavier Jeannin (1997): Expe: An expandable programming language for on-line psychological experiments. Behavior Research Methods 29 (3). Pages 322-327. doi:10.3758/BF03200583 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1344. James Dean Palmer & Eddie Hillenbrand (2009): Reimagining literate programming. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 1007-1014. doi:10.1145/1639950.1640072 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1345. Jens Palsberg & Christina Pavlopoulou (1998): From polyvariant flow information to intersection and union types. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 197-208. doi:10.1145/268946.268963 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1346. Prashant Palvia (1991): On end-user computing productivity: Results of controlled experiments. Information & Management 21 (4). Pages 217-224. doi:10.1016/0378-7206(91)90067-C Exclusion reasons: Q1–2 [III.ajk]Selection was made based on a non-archival full-text copy available at http://libres.uncg.edu/ir/uncg/f/P_Palvia_On_1991.pdf. This article investigates end-user programming using BASIC as the programming language. There was no intent or attempt to evaluate the language, and hence there was no PL design issue.

1347. R. K. Pandey & M. M. Burnett (1993): Is it easier to write matrix manipulation programs visually or textually? An empirical study. In Visual Languages, 1993., Proceedings 1993 IEEE Symposium on. Pages 344-351. doi:10.1109/VL.1993.269621 Exclusion reasons: Q5 [III.ajk]This article empirically compares a visual language to two textual languages. Since visual languages are excluded from our study, that comparison is inadmissible here. Unfortunately, the study was not designed nor analyzed by the authors to compare the two textual languages. The results that can be gleaned from the reported data appear to be inconclusive either way reagrding the two textual languages. As such, I am convinced that this article does not present empirical EVIDENCE regarding any design decision discriminating between the two textual languages.

1348. Raju Pandey & Brant Hashii (1999): Providing Fine-Grained Access Control for Java Programs. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 449-473. doi:10.1007/3-540-48743-3_21 Exclusion reasons: Q1–2 [III.ajk]This article introduces a declarative language for a site to control access to resources. That language does not qualify as a PL in our study, nor is it in any meaningful sense an extension of Java.

1349. John Pane & Brad Myers (1996): Usability issues in the design of novice programming systems. Technical report. http://repository.cmu.edu/isr/820/ Exclusion reasons: Q7 [III.ajk]This article treats the primary studies through their conclusions; the evidence presented in them is not mentioned.

1350. JOHN F. PANE, CHOTIRAT "ANN" RATANAMAHATANA & BRAD A. MYERS (2001): Studying the language and structure in non-programmers' solutions to programming problems. International Journal of Human-Computer Studies 54 (2). Pages 237-264. doi:10.1006/ijhc.2000.0410 Exclusion reasons: Q1–2 [III.ajk]This article reports experiments with human participants designed to elicitate programming styles that come naturally to people. It does not evaluate any language design decisions.

1351. Victor Pankratius, Christoph Schaefer, Ali Jannesari & Walter F. Tichy (2008): Software engineering for multicore systems: an experience report. In Proceedings of the 1st international workshop on Multicore software engineering. New York, NY, USA: ACM. IWMSE '08. Pages 53-60. doi:10.1145/1370082.1370096 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1352. V. Pankratius, A. Jannesari & W.F. Tichy (2009): Parallelizing Bzip2: A Case Study in Multicore Software Engineering. Software, IEEE 26 (6). Pages 70 -77. doi:10.1109/MS.2009.183 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1353. R. J. Parente & H. S. Krasnow (1967): A language for modeling and simulating dynamic systems. Communications of the ACM 10 (9). Pages 559-567. doi:10.1145/363566.363684 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.

1354. Sungwoo Park, Frank Pfenning & Sebastian Thrun (2005): A probabilistic language based upon sampling functions. In Proc. 32nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 171-182. doi:10.1145/1040305.1040320 Exclusion reasons: Q3 [III.ajk]So far as this article may report empirical evidence regarding the efficacy of the design decisions involved, the report is too sparse to evaluate.

1355. Sungwoo Park, Frank Pfenning & Sebastian Thrun (2008): A probabilistic language based on sampling functions. ACM Transactions on Programming Languages and Systems 31 (1). doi:10.1145/1452044.1452048 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1356. C. Park, H. Lee & S. Ryu (2011): An empirical study on the rewritability of the with statement in javascript. In Proc. FOOL'11. http://plrg.kaist.ac.kr/_media/research/publications/fool2011.pdf Exclusion reasons: Q1–2 [II.ajk]This article studies programming language usage patterns to discover if difficult-to-analyze usages are prevalent. It does not speak to construct efficacy.

1357. D. S. Parker (1989): Partial order programming (extended abstract). In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 260-266. doi:10.1145/75277.75300 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1358. David L. Parnas (1966): A language for describing the functions of synchronous systems. Communications of the ACM 9 (2). Pages 72-76. doi:10.1145/365170.365176 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.

1359. David Lorge Parnas (1983): A generalized control structure and its formal definition. Communications of the ACM 26 (8). Pages 572-581. doi:10.1145/358161.358168 Exclusion reasons: Q5 [II.ajk]Formal theoretical work,

1360. David Lorge Parnas (1990): On iterative constructs. ACM Transactions on Programming Languages and Systems 12 (1). Pages 139-141. doi:10.1145/77606.214517 Exclusion reasons: Q5 [III.ajk]This ahort paper does not report an empirical study.

1361. Graham D. Parrington & Santosh K. Shrivastava (1988): Implementing Concurrency Control in Reliable Distributed Object-Oriented Systems. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 233-249. doi:10.1007/3-540-45910-3_14 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1362. Pavel Parízek & OndYej Lhoták (2012): Predicate abstraction of Java programs with collections. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 75-94. doi:10.1145/2384616.2384623 Exclusion reasons: Q1–2 [III.ajk]This program verification method is unlikely to be convertible to a language design choice.

1363. Jacques Pasquier-Boltuck, Ed Grossman & Gérald Collaud (1988): Prototyping an Interactive Electronic Book System Using an Object-Oriented Approach. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 177-190. doi:10.1007/3-540-45910-3_11 Exclusion reasons: Q1–2 [II.ajk]No language comparison, no PL design issue.

1364. MUKESH J. PATEL, BENEDICT DU BOULAY & CHRIS TAYLOR (1997): Comparison of contrasting Prolog trace output formats. International Journal of Human-Computer Studies 47 (2). Pages 289-322. doi:10.1006/ijhc.1997.0119 Exclusion reasons: Q1–2 [II.ajk]No language design issues.

1365. J. B. Paterson (1963): The COBOL sorting verb. Communications of the ACM 6 (5). Pages 255-258. doi:10.1145/366552.366588 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1366. Basawaraj Patil, Klaus Maetzel & Erich J. Neuhold (2001): Native-End User Languages: A Design Framework. In PPIG 2001. (Found in http://ppig.org/workshops/13th-programme.html.) Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.

1367. David A. Patterson & Richard S. Piepho (1982): RISC assessment: A high-level language experiment. SIGARCH Comput. Archit. News 10 (3). Pages 3-8. http://dl.acm.org/citation.cfm?id=1067649.801708 Exclusion reasons: Q1–2 [II.ajk]Studies machines, not languages.

1368. Jeeva Paudel & José Nelson Amaral (2011): Using the Cowichan problems to investigate the programmability of X10 programming system. In Proceedings of the 2011 ACM SIGPLAN X10 Workshop. New York, NY, USA: ACM. X10 '11. Article 4. Pages 4:1-4:10. doi:10.1145/2212736.2212740 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions (no comparison).

1369. Sebastian Pavel, Jacques Noyé & Jean-Claude Royer (2004): Dynamic Configuration of Software Product Lines in ArchJava. Volume 3154.In Nord, Robert (ed.) Software Product Lines.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 111-113. doi:10.1007/978-3-540-28630-1_6 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions (no comparison).

1370. S. J. Payne, M. E. Sime & T. R. G. Green (1984): Perceptual structure cueing in a simple command language. International Journal of Man-Machine Studies 21 (1). Pages 19-29. doi:10.1016/S0020-7373(84)80036-X Exclusion reasons: Q1–2 [III.ajk]The language under discussion is not a programming language.

1371. Stephen J. Payne & T. R. G. Green (1989): The structure of command languages: an experiment on task-action grammar. International Journal of Man-Machine Studies 30 (2). Pages 213-234. doi:10.1016/S0020-7373(89)80011-2 Exclusion reasons: Q1–2 [III.ajk]This article reports a controlled experiment with human participants that (stated simplistically) studies how a command language's syntax regularity affectes learnability and usability. This may have some application to programming language design but the connection is too remote for our purposes.

1372. Cristian Perfumo, Nehir Sönmez, Srdjan Stipic, Osman Unsal, Adrián Cristal, Tim Harris & Mateo Valero (2008): The limits of software transactional memory (STM): dissecting Haskell STM applications on a many-core environment. In Proceedings of the 5th conference on Computing frontiers. New York, NY, USA: ACM. CF '08. Pages 67–78. doi:10.1145/1366230.1366241 Exclusion reasons: Q1–2 [III.ajk]This article deals with implementation issues only,

1373. A. J. Perlis & K. Samelson (1958): Preliminary report: international algebraic language. Communications of the ACM 1 (12). Pages 8-22. doi:10.1145/377924.594925 Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative study.

1374. A. J. Perlis & Renato Iturriaga (1964): An extension to ALGOL for manipulating formulae. Communications of the ACM 7 (2). Pages 127-130. doi:

10.1145/363921.363943 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1375. A. J. Perlis (1964): A format language. Communications of the ACM 7 (2). Pages 89-97. doi:10.1145/363921.363936 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1376. Gary PerlmanI (1984): Natural artificial languages: low level processes. International Journal of Man-Machine Studies 20 (4). Pages 373-419. doi: 10.1016/S0020-7373(84)80075-9 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
1377. R. H. Perrott (1979): A Language for Array and Vector Processors. ACM Transactions on Programming Languages and Systems 1 (2). Pages 177-195. doi:10.1145/357073.357075 Exclusion reasons: Q1–2 [II.ajk]No comparison; no empirics.
1378. R. H. Perrott & P. S. Dhillon (1981): An experiment with Fortran and Pascal. Software: Practice and Experience 11 (5). Pages 491-496. doi:10.1002/spe.4380110507 Exclusion reasons: Q1–2 [III.ajk]This article reports on a study in which a program written in Fortran is compared with a manual translation of it to Pascal with respect to performance, using particular implementations of each language. Thus, it does not present a PL design issue.
1379. R. H. Perrott & A. Ramasubbu (1992): An experiment in concurrent software evaluation. Concurrency: Practice and Experience 4 (7). Pages 533-555. doi:10.1002/cpe.4330040704 Exclusion reasons: Q5 [III.ajk]Evaluation is purely analytical in nature.
1380. Tom Di Persio, Dan Isbister & Ben Shneiderman (1980): An experiment using memorization/reconstruction as a measure of programmer ability. International Journal of Man-Machine Studies 13 (3). Pages 339-354. doi:10.1016/S0020-7373(80)80047-2 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1381. W. W. Peterson, T. Kasami & N. Tokura (1973): On the capabilities of while, repeat, and exit statements. Communications of the ACM 16 (8). Pages 503-512. doi:10.1145/355609.362337 Exclusion reasons: Q1–2 [II.ajk]Theoretical work.
1382. Giles Peterson & Aaron B. Budgor (1980): The computer language Mathsy and applications to solid state physics. Communications of the ACM 23 (8). Pages 466-474. doi:10.1145/358896.358900 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1383. Leons Petrazickis (2009): Deploying PHP applications on IBM DB2 in the cloud: MediaWiki as a case study. In Proceedings of the 2009 Conference of the Center for Advanced Studies on Collaborative Research. New York, NY, USA: ACM. Pages 304-305. doi:10.1145/1723028.1723069 Exclusion reasons: Q1–2 [II.ajk]No comparison, at least.
1384. L. Petrone & C. E. Vandoni (1964): Integer and signed constants in ALGOL. Communications of the ACM 7 (12). Pages 734-735. doi:10.1145/355588.365138 Exclusion reasons: Q1–2 [II.ajk]Studies language specification.
1385. Simon L. Peyton Jones & Philip Wadler (1993): Imperative functional programming. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 71-84. doi:10.1145/158511.158524 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1386. Simon Peyton Jones, Andrew Gordon & Sigbjorn Finne (1996): Concurrent Haskell. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 295-308. doi:10.1145/237721.237794 Exclusion reasons: Q5 [III.ajk]This article, evaluated based on the version at http://www.haskell.org/ghc/docs/papers/concurrent-haskell.ps.gz, does not aspire to empiricity.
1387. John Pfaltz (2000): Data Providers — A Language Experiment. Volume 1966.In Bhalla, Subhash (ed.) Databases in Networked Information Systems. Lecture Notes in Computer Science. Pages 33-44. doi:10.1007/3-540-44431-9_3 Exclusion reasons: Q5 [III.ajk]This article, though styling itself as reporting an "experiment", does not aspire to empiricality.
1388. Jr. Pfeiffer, J.J. (1998): Case study: developing a rule-based language for mobile robots. In Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on. Pages 204-209. doi:10.1109/VL.1998.706164 Exclusion reasons: Q1–2 [III.ajk]Visual languages are excluded.
1389. J. Richard Phillips & H. C. Adams (1972): Dynamic partitioning for array languages. Communications of the ACM 15 (12). Pages 1023-1032. doi: 10.1145/361598.361606 Exclusion reasons: Q5 [III.ajk]This constructive-analytical article has no aspiration for empiricity.
1390. Andrew Phillips (2009): @composite: macro annotations for Java C. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 767-768. doi:10.1145/1639950.1640005 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.
1391. Scott M. Pike, Wayne D. Heym, Bruce Adcock, Derek Bronish, Jason Kirschenbaum & Bruce W. Weide (2009): Traditional assignment considered harmful. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 909-916. doi:10.1145/1639950.1640056 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1392. Nicholas Pippenger (1996): Pure versus impure Lisp. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 104-109. doi:10.1145/237721.237741 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1393. Nicholas Pippenger (1997): Pure versus impure Lisp. ACM Transactions on Programming Languages and Systems 19 (2). Pages 223-238. doi:10.1145/244795.244798 Exclusion reasons: Q5 [III.ajk]This theoretical article does not aspire to empiricity.
1394. Laleh Pirzadeh (2010): Human Factors in Software Development: A Systematic Literature Review. . A Master's Thesis at Chalmers University of Technology. http://publications.lib.chalmers.se/cpl/record/index.xsql?pubid=126748 Exclusion reasons: Q1–2 [II.ajk]No PL design issue focus.
1395. Andrew M. Pitts & Mark R. Shinwell (2007): Generative unbinding of names. In Proc. 34th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 85-95. doi:10.1145/1190216.1190232 Exclusion reasons: Q5 [II.ajk]Formal type-theoretical work.
1396. Filip Pizlo & Jan Vitek (2006): An Emprical Evaluation of Memory Management Alternatives for Real-Time Java. In Real-Time Systems Symposium, 2006. RTSS '06. 27th IEEE International. Pages 35-46. doi:10.1109/RTSS.2006.9 Exclusion reasons: Q1–2 [III.ajk]This article evaluates implementation techniques only.
1397. Hans-Jürgen Plewan & Peter Schlenk (1990): Creating and controlling concurrency in object oriented systems — A case study. Volume 457.In Burkhart, Helmar (ed.) CONPAR 90 — VAPP IV.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 616-627. doi:10.1007/3-540-53065-7_138 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.
1398. Amnart Pohthong & David Budgen (2001): Reuse strategies in software development: an empirical study. Information and Software Technology 43 (9). Pages 561-575. doi:10.1016/S0950-5849(01)00166-5 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design issues.
1399. J. Gary Polhill, Lee-Ann Sutherland & Nicholas M. Gotts (2010): Using Qualitative Evidence to Enhance an Agent-Based Modelling System for Studying Land Use Change. Journal of Artificial Societies and Social Simulation 13 (2). Pages 10. http://jasss.soc.surrey.ac.uk/13/2/10.html Exclusion reasons: Q1–2 [II.ajk]This article has no language design relevance.
1400. Gerardo Cepeda Porras & Yann-Gaël Guéhéneuc (2010): An empirical study on the efficiency of different design pattern representations in UML class diagrams. Empirical Software Engineering 15 (5). Pages 493-522. doi:10.1007/s10664-009-9125-9 Exclusion reasons: Q1–2 [II.ajk]Graphical languages are excluded from our definition of programming languages.
1401. Daryl Posnett, Christian Bird & Prem Dévanbu (2011): An empirical study on the influence of pattern roles on change-proneness. Empirical Software Engineering 16 (3). Pages 396-423. doi:10.1007/s10664-010-9148-2 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1402. Matthew Powers, Conda Lashley, Pamela Sanchez & Ben Shneiderman (1984): An experimental comparison of tabular and graphic data presentation. International Journal of Man-Machine Studies 20 (6). Pages 545-566. doi:10.1016/S0020-7373(84)80029-2 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1403. Polyvios Pratikakis, Jaime Spacco & Michael Hicks (2004): Transparent proxies for java futures. In OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 206-223. doi:10.1145/1028976.1028994 Exclusion reasons: Q5 [III.ajk]This article's evaluation is analytic or anecdotal, not empirical.
1404. T. W. Pratt & Robert K. Lindsay (1966): A Processor-Building System for Experimental Programming Languages. In AFIPS 1966 Proceedings of the Fall Joint Computer Conference. Pages 613. doi:10.1145/1464291.1464358 Exclusion reasons: Q1–2 [III.ajk]Discusses implementation techniques.
1405. Terrence W. Pratt & Daniel P. Friedman (1971): A language extension for graph processing and its formal semantics. Communications of the ACM 14 (7). Pages 460-467. doi:10.1145/362619.362627 Exclusion reasons: Q1–2 [II.ajk]Development of a language feature, and discussion of a feature development process; no evaluation based on the abstract.
1406. Vaughan R. Pratt (1977): The competence/performance dichotomy in programming preliminary report. In Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 194-200. doi:10.1145/512950.512968 Exclusion reasons: Q1–2 [II.ajk]Development of a new programming approach.
1407. T.W. Pratt, D.E. Brown, T. Flory, G.D. Maydwell, J. McCauley, B. Murill, F. Powell, M. Tucker, R. Wayland & J. Wilson (1979): Val: an experiment in programming language design and definition. Technical report. (publication data unavailable at this time.) Exclusion reasons: Q1–2 [III.ajk]Language definition with no evaluation.
1408. Vaughan Pratt (1983): Five paradigm shifts in programming language design and their realization in Viron, a dataflow programming environment. In Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 1-9. doi:10.1145/567067.567068 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1409. Vaughn Pratt (1991): Modeling concurrency with geometry. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 311-322. doi:10.1145/99583.99625 Exclusion reasons: Q1–2 [III.ajk]This is an automata-theoretical paper.

1410. David R. Pratt, Anthony J. Courtemanche, Jamie Moyers & Charles Campbell (2000): An empirical evaluation of the Java and C++ programming languages. In Proc. The Interservice/Industry Training, Simulation & Education Conference (I/ITSEC) 2000. http://ntsa.metapress.com/link.asp?id=cvmg9cuu7cjklu63 Exclusion reasons: Q1–2Q3 [III.ajk]This article reports a study in which several programs were written in Java and C++, and the programs' performance was compared. It is not clear from the article how exactly the Java and C++ versions are related to each other, but it appears they are not independent reimplementations. It appears that the study is more about comparing implementations than language designs.

1411. Lutz Prechelt (2001): Kontrollierte Experimente in der Softwaretechnik. Springer. Exclusion reasons: Q4 [II.ajk]Auf Deutsch

1412. W Pree & G Pomberger (1992): Object-oriented versus conventional software development: A comparative case study. Microprocessing and Microprogramming 35 (1-5). Pages 203-211. doi:10.1016/0165-6074(92)90318-2 Exclusion reasons: Q1–2Q3 [III.ajk]This article reports a study where two programs, one written in Modula-2 and the other in C++, are compared analytically and numerically in order to compare conventional, "module-oriented" and object-oriented software development. It is doubtful whether this study can be interpreted as evaluating a PL design decision. Moreover, the article does not adequately explain where the two programs were acquired, who built them and whether this building was a part of this study instead of being existing software. Taken together, these considerations convince me that the article should be excluded.

1413. Christian Prehofer (1997): Feature-oriented programming: A fresh look at objects. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 419-443. doi:10.1007/BFb0053389 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1414. David E Price (2007): Using a 'Wizard of Oz' Study to Evaluate a Spoken Language Interface for Programming. . MSc Thesis at The University of Utah. http://www.cs.utah.edu/~riloff/pdfs/Price-MS-Thesis.pdf Exclusion reasons: Q1–2 [III.ajk]This thesis evaluates a programming interface, not any language design decisions.

1415. David Price, Ellen Riloff & Joseph Zachary (2007): A Study to Evaluate a Natural Language Interface for Computer Science Education. In Proc. AIED 2007. http://www.cs.utah.edu/~riloff/pdfs/aied-wkshp07.pdf Exclusion reasons: Q1–2 [III.ajk]This article deals with a natural-language interface to programming in Java. It seems to me fair to regard it as a user interface rather than a programming language by virtue of the way it is used in the study. In any case, there is no comparison and thus no design decision at play.

1416. Todd A. Proebsting & Scott A. Watterson (1996): Filter fusion. In Proc. 23rd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 119-130. doi:10.1145/237721.237760 Exclusion reasons: Q1–2 [III.ajk]This article presents and evaluates a new compiler optimization method. There is no PL design issue here.

1417. N. S. Prywes, A. Pnueli & S. Shastry (1979): Use of a Nonprocedural Specification Language and Associated Program Generator in Software Development. ACM Transactions on Programming Languages and Systems 1 (2). Pages 196-217. doi:10.1145/357073.357076 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1418. A. Przybylek (2011): Impact of Aspect-Oriented Programming on Software Modularity. In Software Maintenance and Reengineering (CSMR), 2011 15th European Conference on. Pages 369-372. doi:10.1109/CSMR.2011.55 Exclusion reasons: Q1–2 [III.ajk]This article does not report a completed study.

1419. J. Pugh & D. Simpson (1979): Pascal errors–empirical evidence. Computer Bulletin (March). Pages 26-28. Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions (no comparison).

1420. W. Pugh & T. Teitelbaum (1989): Incremental computation via function caching. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 315-328. doi:10.1145/75277.75305 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1421. Helen C. Purchase, Ray Welland, Matthew McGill & Linda Colpoys (2004): Comprehension of diagram syntax: an empirical study of entity relationship notations. International Journal of Human-Computer Studies 61 (2). Pages 187-203. doi:10.1016/j.ijhcs.2004.01.003 Exclusion reasons: Q1–2 [II.ajk]The notations aren't textual (as required by our PL definition)

1422. James M. Purtilo (1994): The POLYLITH software bus. ACM Transactions on Programming Languages and Systems 16 (1). Pages 151-174. doi:10.1145/174625.174629 Exclusion reasons: Q1–2 [II.ajk]No PL design issues.

1423. O. Pustovalova & U. Montanari (2012): Constraint Logic Programming for Service-Oriented Computing: A Case Study in Prova. Technical report. http://www.imtlucca.it/_documents/other_files/007312_0_Technical_Report_-_Constraint_Logic_Programming_for_Service-Oriented_Computing._A_Case_Study_in_Prova.pdf Exclusion reasons: Q1–2 [II.ajk]No language design decision involved.

1424. Ian Pye (2011): Locks, deadlocks and abstractions: experiences with multi-threaded programming at CloudFlare, Inc.. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 129-132. doi:10.1145/2095050.2095073 Exclusion reasons: Q1–2 [III.ajk]This article does not discuss language design issues.

1425. Hari K. Pyla, Calvin Ribbens & Srinidhi Varadarajan (2011): Exploiting coarse-grain speculative parallelism. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 555-574. doi:10.1145/2048066.2048110 Exclusion reasons: Q1–2 [III.ajk]Evaluation does not compare to alternative choices.

1426. I. C. Pyle (1962): Character manipulation in FORTRAN. Communications of the ACM 5 (8). Pages 432-433. doi:10.1145/368637.368650 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1427. I. C. Pyle (1963): Dialects of FORTRAN. Communications of the ACM 6 (8). Pages 462-467. doi:10.1145/366707.367586 Exclusion reasons: Q1–2 [III.ajk]This article is an analytic review of early FORTRAN dialects and their differences; there is no issue of efficacy.

1428. Xin Qi & Andrew C. Myers (2009): Masked types for sound object initialization. In Proc. 36th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 53-65. doi:10.1145/1480881.1480890 Exclusion reasons: Q5 Disagreement resolution result. [III.ajk]The evaluation in this article, so far as it may be empirical in nature, is not reported in such a way that the empirical validity of the evaluation could be assessed based on this report. However, there's a good case against considering the evaluation empirical, as it is primarily an experience report with little systematicality to it. [sel-2.kaijanaho]Experience report, no other potential empiricity. [sel-2.tirronen]Presents only a personal experience report

1429. Xiaolei Qian & Allen Goldberg (1993): Referential opacity in nondeterministic data refinement. ACM Transactions on Programming Languages and Systems 2 (1-4). Pages 233-241. doi:10.1145/176454.176578 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1430. Zhenyu Qian (1994): Higher-order equational logic programming. In Proc. 21th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 254-267. doi:10.1145/174675.177889 Exclusion reasons: Q1–2 [II.ajk]Implementation discussion, theoretical study.

1431. Christian Queinnec & Bernard Serpette (1991): A dynamic extent control operator for partial continuations. In Proc. 18th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 174-184. doi:10.1145/99583.99610 Exclusion reasons: Q5 [III.ajk]This article approaches its topic analytically, with no empiricality evident.

1432. Jaime Quinonez, Matthew S. Tschantz & Michael D. Ernst (2008): Inference of Reference Immutability. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 616-641. doi:10.1007/978-3-540-70592-5_26 Exclusion reasons: Q1–2 [II.ajk]Program analysis technique.

1433. Irving N. Rabinowitz (1962): Report on the algorithmic language FORTRAN II. Communications of the ACM 5 (6). Pages 327-337. doi:10.1145/367766.368151 Exclusion reasons: Q1–2 [III.ajk]This article is a language definition.

1434. George Radin & H. Paul Rogoway (1965): NPL: highlights of a new programming language. Communications of the ACM 8 (1). Pages 9-17. doi:10.1145/363707.363708 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition.

1435. Hridesh Rajan & Gary T. Leavens (2008): Ptolemy: A Language with Quantified, Typed Events. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 155-179. doi:10.1007/978-3-540-70592-5_8 Exclusion reasons: Q3 [III.ajk]This article introduces and analyzes a new language. It also presents a comparison of the language with related languages; this comparison might be considered empirical but in the absence of a description of the methodology used in the data collection and analysis, the report cannot be considered complete as to the potentially empirical aspect of the work; no other report of the same work has been identified.

1436. Rajeev R. Raje, Ming Zhong & Tongyu Wang (2001): Case study: a distributed concurrent system with AspectJ. SIGAPP Applied Computing Review 9 (2). Pages 17-23. doi:10.1145/512000.512004 Exclusion reasons: Q1–2Q3 [III.ajk]It isn't clear whether this article reports a comparative evaluation or not; to the extent it does, it is insufficiently described.

1437. Vaclav Rajlich & Shivkumar Ragunathan (1998): A case study of evolution in object oriented and heterogeneous architectures. Journal of Systems and Software 43 (2). Pages 85-91. doi:10.1016/S0164-1212(98)10024-9 Exclusion reasons: Q1–2 [III.ajk]This article deals with architecture, not language design decisions.

1438. Raghu Ramakrishnan & Abraham Silberschatz (1986): Annotations for distributed programming in logic. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 255-262. doi:10.1145/512644.512668 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1439. Raghu Ramakrishnan (1990): Parallelism in logic programs. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 246-260. doi:10.1145/96709.96734 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1440. Allan Ramsay (1984): Type-checking in an untyped language. International Journal of Man-Machine Studies 20 (2). Pages 157-167. doi:10.1016/

S0020-7373(84)80015-2 Exclusion reasons: Q5 [II.ajk]No empirical validation, at least according to the abstract.

1441. H. Rudy Ramsey, Michael E. Atwood & James R. Van Doren (1983): Flowcharts versus program design languages: an experimental comparison. Communications of the ACM 26 (6). Pages 445-449. doi:10.1145/358141.358149 Exclusion reasons: Q1–2 [II.ajk]Not programming languages by our definition.

1442. Bertram Raphael (1966): The structure of programming languages. Communications of the ACM 9 (2). Pages 67-71. doi:10.1145/365170.365175 Exclusion reasons: Q1–2 [II.ajk]Analysis of the programming language concept.

1443. Bertram Raphael (1966): The structure of programming languages. Communications of the ACM 9 (3). Pages 155-156. doi:10.1145/365230.365255 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1444. A. Rashid, T. Cottenier, P. Greenwood, R. Chitchyan, R. Meunier, R. Coelho, M. Sudholt & W. Joosen (2010): Aspect-Oriented Software Development in Practice: Tales from AOSD-Europe. Computer 43 (2). Pages 19-26. doi:10.1109/MC.2010.30 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate a language design decision.

1445. Aseem Rastogi, Avik Chaudhuri & Basil Hosmer (2012): The ins and outs of gradual type inference. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 481-494. doi:10.1145/2103656.2103714 Exclusion reasons: Q1–2 [III.ajk]The evaluation in this article focuses on whether the algorithm can recover enough omitted type information not to lose too much in performance. This is not an efficacy issue.

1446. J. Raymond (1976): LG: a language for analytic geometry. Communications of the ACM 19 (4). Pages 182-187. doi:10.1145/360032.360042 Exclusion reasons: Q5 [III.ajk]This language exposition does not aspire to empiricity.

1447. T. P. Reagan, G. J. Vecellio, W. Battle, A. M. Englehart, R. H. Paris & N. Stewart (1990): An Ada software port case study. In Proceedings of the Ada-Europe international conference on Ada : experiences and prospects: experiences and prospects. New York, NY, USA: Cambridge University Press. Pages 348–360. http://dl.acm.org/citation.cfm?id=103367.103653 Exclusion reasons: Q1–2 [III.ajk]This article does not appear to evaluate the language design.

1448. David R. Reed, Marty Cagan, Ted Goldstein & Barbara Moo (1991): Issues in moving from C to C++. In OOPSLA '91: Conference proceedings on Object-oriented programming systems, languages, and applications. Pages 163-165. doi:10.1145/117954.117966 Exclusion reasons: Q1–2 [III.ajk]This article does not report a study.

1449. Paul Reed (2000): Building Your Own Tools: An Oberon Industrial Case-Study. Volume 1897.In Weck, Wolfgang and Gutknecht, Jürg (ed.) Modular Programming Languages.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 291-298. doi:10.1007/10722581_23 Exclusion reasons: Q1–2 [II.ajk]No comparison language.

1450. Eric C. Reed, Nicholas Chen & Ralph E. Johnson (2011): Expressing pipeline parallelism using TBB constructs: a case study on what works and what doesn't. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 133-138. doi:10.1145/2095050.2095074 Exclusion reasons: Q1–2 [III.ajk]Evaluation only by performance.

1451. T. Reenskaug & A. L. Skaar (1989): An environment for literate Smalltalk programming. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 337-345. doi:10.1145/74877.74912 Exclusion reasons: Q1–2 [II.ajk]Not a language by our definition.

1452. E. D. Reilly, Jr. & F. D. Federighi (1965): On reversible subroutines and computers that run backwards. Communications of the ACM 8 (9). Pages 557-558. doi:10.1145/365559.365593 Exclusion reasons: Q1–2 [II.ajk]Discusses a computational model and its implications; no direct PL design issue.

1453. Gabriel Dos Reis & Bjarne Stroustrup (2006): Specifying C++ concepts. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 295-308. doi:10.1145/1111037.1111064 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

1454. Martin Rem (1981): Associons: A Program Notation with Tuples Instead of Variables. ACM Transactions on Programming Languages and Systems 3 (3). Pages 251-262. doi:10.1145/357139.357142 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.

1455. Bin Ren, Gagan Agrawal, Brad Chamberlain & Steve Deitz (2011): Translating Chapel to Use FREERIDE: A Case Study in Using an HPC Language for Data-Intensive Computing. In Parallel and Distributed Processing Workshops and Phd Forum (IPDPSW), 2011 IEEE International Symposium on. Pages 1242-1249. doi:10.1109/IPDPS.2011.266 Exclusion reasons: Q1–2 [III.ajk]This article evaluates an implementation approach, not any language design decision.

1456. John Reppy & Aaron Turon (2007): Metaprogramming with Traits. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 373-398. doi:10.1007/978-3-540-73589-2_18 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.

1457. Jennifer Rexford (2012): Programming languages for programmable networks. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 215-216. doi:10.1145/2103656.2103683 Exclusion reasons: Q3 [III.ajk]This is an abstract and talk only.

1458. John C. Reynolds (1970): GEDANKEN - a simple typeless language based on the principle of completeness and the reference concept. Communications of the ACM 13 (5). Pages 308-319. doi:10.1145/362349.362364 Exclusion reasons: Q1–2 [II.ajk]No comparison.

1459. John C. Reynolds (1978): Syntactic control of interference. In Proc. 5th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 39-46. doi:10.1145/512760.512766 Exclusion reasons: Q5 [III.ajk]This article is an analytical study of a PL design issue; it has no aspiration to empiricity.

1460. Gregor Richards, Sylvain Lebresne, Brian Burg & Jan Vitek (2010): An analysis of the dynamic behavior of JavaScript programs. In Proceedings of the 2010 ACM SIGPLAN conference on Programming language design and implementation. New York, NY, USA: ACM. PLDI '10. Pages 1-12. doi:10.1145/1806596.1806598 Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study of Javascipt programs in the wild. There is no language design decision at play.

1461. Gregor Richards, Christian Hammer, Brian Burg & Jan Vitek (2011): The Eval That Men Do: A Large-Scale Study of the Use of Eval in JavaScript Applications. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 52-78. doi:10.1007/978-3-642-22655-7_4 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1462. Joel E. Richardson, Michael J. Carey & Daniel T. Schuh (1993): The design of the E programming language. ACM Transactions on Programming Languages and Systems 15 (3). Pages 494-534. doi:10.1145/169683.174157 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1463. James Riely & Matthew Hennessy (1998): A typed language for distributed mobile processes (extended abstract). In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 378-390. doi:10.1145/268946.268978 Exclusion reasons: Q5 [III.ajk]This paper does not aspire to empiricity.

1464. M. van Riemsdijk & Koen Hindriks (2009): An Empirical Study of Agent Programs. Volume 5925.In Yang, Jung-Jin and Yokoo, Makoto and Ito, Takayuki and Jin, Zhi and Scerri, Paul (ed.) Principles of Practice in Multi-Agent Systems. Lecture Notes in Computer Science. Pages 200-215. doi:10.1007/978-3-642-11161-7_14 Exclusion reasons: Q1–2 [III.ajk]This article is an exploratory study of programming style in the language GOAL. It does not present a PL design issue.

1465. Martin C. Rinard & Monica S. Lam (1998): The design, implementation, and evaluation of Jade. ACM Transactions on Programming Languages and Systems 20 (3). Pages 483-545. doi:10.1145/291889.291893 Exclusion reasons: Q5 [III.ajk]This article includes a section where several existing programs are manually converted into the new language design. Their static metrics are compared and the new programs' performance is measured. This is more a feasibility evaluation than an efficacy one.

1466. Ran Rinat & Menachem Magidor (1996): Metaphoric polymorphism: Taking code reuse one step further. In Proc. ECOOP'96 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1098. Pages 449-471. doi:10.1007/BFb0053073 Exclusion reasons: Q5 [III.ajk]This analytical-constructive paper does not aspire to empiricity.

1467. Ran Rinat & Scott Smith (2002): Modular Internet Programming with Cells. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 257-280. doi:10.1007/3-540-47993-7_12 Exclusion reasons: Q1–2 [II.ajk]Language exposition

1468. G. David Ripley & Frederick C. Druseikis (1978): A statistical analysis of syntax errors. Computer Languages 3 (4). Pages 227-240. doi:10.1016/0096-0551(78)90041-3 Exclusion reasons: Q1–2 [II.ajk]This article studies language misuse patterns for a single language; there is no language design issue in focus.

1469. Peter Ripota (1974): A concept for a primary author's language (PAL). International Journal of Man-Machine Studies 6 (4). Pages 465-478. doi:10.1016/S0020-7373(74)80014-3 Exclusion reasons: Q1–2 [II.ajk]This sounds like an intermediate language, and in any case there's no empirical evaluation.

1470. Frederic N. Ris (1984): Experience with access functions in an experimental compiler. IBM Journal of Research and Development 28 (1). Pages 40-51. doi:10.1147/rd.281.0040 Exclusion reasons: Q5 [III.ajk]So far as this article evaluates a language decision, it does it by exploring its implications (whether something can be done in it), which is non-empirical by our standards.

1471. L.S. Rising & F.W. Calliss (1993): An experiment investigating the effect of information hiding on maintainability. In Computers and Communications, 1993., Twelfth Annual International Phoenix Conference on. Pages 510-516. doi:10.1109/PCCC.1993.344523 Exclusion reasons: Q1–2 [II.ajk]Studies programming styles.

1472. Chris Roast (2002): Dimension Driven Re-Design - Applying Systematic Dimensional Analysis. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q1–2Q5 [III.ajk]This article does not evaluate a language design decision.

1473. D. K. Robbins (1962): FORTRAN for business data processing. Communications of the ACM 5 (7). Pages 412-414. doi:10.1145/368273.368582 Exclusion reasons: Q5 [III.ajk]This article is mostly a language exposition. The few empirical claims are anecdotal, with no supporting data nor any indication that such has been collected.

1474. Scott P. Robertson & Chiung-Chen Yu (1990): Common cognitive representations of program code across tasks and languages. International Journal of Man-Machine Studies 33 (3). Pages 343-360. doi:10.1016/S0020-7373(05)80123-3 Exclusion reasons: Q1–2 [II.ajk]Theory of programmer cognition, no language design issue.

1475. Pierre N. Robillard (1986): Schematic pseudocode for program constructs and its computer automation by SCHEMACODE. Communications of the ACM 29 (11). Pages 1072-1089. doi:10.1145/7538.7541 Exclusion reasons: Q1–2 [III.ajk]This system exposition article does not aspire to empiricity.

1476. Martin Robillard & Robert DeLine (2011): A field study of API learning obstacles. Empirical Software Engineering 16 (6). Pages 703-732. doi:10.1007/s10664-010-9150-8 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

1477. Natalia Romero, María José Presso, Verónica Argañaraz, Gabriel Baum & Máximo Prieto (1998): Purpose: Between Types and Code. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 588. doi:10.1007/3-540-49255-0_14 Exclusion reasons: Q5 [III.ajk]This super-short article has no aspiration to empiricity.

1478. Pablo Romero (1999): Focal structures in Prolog. In PPIG 1999. (Found in http://ppig.org/workshops/11th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

1479. PABLO ROMERO (2001): Focal structures and information types in Prolog. International Journal of Human-Computer Studies 54 (2). Pages 211-236. doi:10.1006/ijhc.2000.0408 Exclusion reasons: Q1–2 [II.ajk]This article evaluates empirically a theory of program comprehension.

1480. P. Romero & B. du Boulay (2004): Structural Knowledge and Language Notational Properties in Program Comprehension. In Visual Languages and Human Centric Computing, 2004 IEEE Symposium on. Pages 223-225. doi:10.1109/VLHCC.2004.50 Exclusion reasons: Q1–2 [II.ajk]No comparison; no design issue.

1481. Patrick Maxim Rondon, Ming Kawaguchi & Ranjit Jhala (2010): Low-level liquid types. In Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 131-144. doi:10.1145/1706299.1706316 Exclusion reasons: Q3 Q5 [III.ajk]This analytical-constructive article includes a brief evaluation section. However, it appears to evaluate whether the technique is good enough (can it do what it's supposed to?) and is thus analytic, not empirical. The methodology is also described fairly vaguely.

1482. Gene F. Rose (1964): An extension of ALGOL-like languages. Communications of the ACM 7 (2). Pages 52-61. doi:10.1145/363921.363925 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1483. Barry K. Rosen (1977): Applications of high level control flow. In Proc. 4th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 38-47. doi:10.1145/512950.512955 Exclusion reasons: Q1–2 [II.ajk]Program analysis technique, of a sort. No PL design issue.

1484. M. B. Rosson & E. Gold (1989): Problem-solution mapping in object-oriented design. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 7-10. doi:10.1145/74877.74880 Exclusion reasons: Q1–2 [III.ajk]This study does not evaluate a language design decision.

1485. Mary Beth Rosson, John M. Carrol & Rachel K. E. Bellamy (1990): Smalltalk scaffolding: a case study of minimalist instruction. In Proceedings of the SIGCHI conference on Human factors in computing systems: Empowering people. New York, NY, USA: ACM. Pages 423-430. doi:10.1145/97243.97319 Exclusion reasons: Q1–2 [II.ajk]Teaching, not PL design, relevant.

1486. Mary Beth Rosson & John M. Carroll (1993): Active Programming Strategies in Reuse. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 4-20. doi:10.1007/3-540-47910-4_2 Exclusion reasons: Q1–2 [II.ajk]Study of programmer activity.

1487. Mary Beth Rosson & John M. Carroll (1996): The reuse of uses in Smalltalk programming. ACM Transactions on Computer-Human Interaction 3 (3). Pages 219-253. doi:10.1145/234526.234530 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1488. Christopher E. Rothe (1981): An abstract programming model. Communications of the ACM 24 (9). Pages 594-596. doi:10.1145/358746.358766 Exclusion reasons: Q1–2 [II.ajk]No PL design issue studied.

1489. Francois Rouaix (1990): Safe run-time overloading. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 355-366. doi:10.1145/96709.96746 Exclusion reasons: Q5 [II.ajk]Based on the abstract, formal development only.

1490. Peter Van Roy, Seif Haridi, Per Brand, Gert Smolka, Michael Mehl & Ralf Scheidhauer (1997): Mobile objects in distributed Oz. ACM Transactions on Programming Languages and Systems 19 (5). Pages 804-851. doi:10.1145/265943.265972 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1491. R. J. Rubey (1968): A comparative evaluation of PL/I. Datamation 14 (12). Pages 22-25. Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical comparison of PL/I with several other languages. The design of the study is such that it doesn't evaluate any particular language design decisions.

1492. Salvatore Ruggieri & Fred Mesnard (2010): Typing linear constraints. ACM Transactions on Programming Languages and Systems 32 (6). doi:10.1145/1749608.1749610 Exclusion reasons: Q1–2 [II.ajk]Type-theoretical and implementation study.

1493. James Rumbaugh (1987): Relations as semantic constructs in an object-oriented language. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 466-481. doi:10.1145/38765.38850 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1494. James Rumbaugh (1988): Controlling propagation of operations using attributes on relations. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 285-296. doi:10.1145/62083.62109 Exclusion reasons: Q5 [III.ajk]This paper has no aspiration to empiricity.

1495. Colin Runciman (1981): Modula and a vision laboratory. International Journal of Man-Machine Studies 14 (3). Pages 371-386. doi:10.1016/S0020-7373(81)80064-8 Exclusion reasons: Q5 [II.ajk]Analytical in approach.

1496. Chandan R. Rupakheti & Daqing Hou (2008): An empirical study of the design and implementation of object equality in Java. In Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds. Pages 9:111–9:125. doi:10.1145/1463788.1463800 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1497. Robert D. Russell (1978): The PDP-11: A case study of how not to design condition codes. In Proceedings of the 5th annual symposium on Computer architecture. New York, NY, USA: ACM. Pages 190-194. doi:10.1145/800094.803047 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.

1498. Vincent Russo, Gary Johnston & Roy Campbell (1988): Process management and exception handling in multiprocessor operating systems using object-oriented design techniques. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 248-258. doi:10.1145/62083.62105 Exclusion reasons: Q5 [III.ajk]So far as this article evaluates a language design decision, which is not by any means clear, it does that analytically, not empirically.

1499. Claudio V. Russo (2008): Join patterns for visual basic. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 53-72. doi:10.1145/1449764.1449770 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1500. Barbara G. Ryder, Mary Lou Soffa & Margaret Burnett (2005): The impact of software engineering research on modern progamming languages. ACM Transactions on Software Engineering and Methodology 14 (4). Pages 431-477. doi:10.1145/1101815.1101818 Exclusion reasons: Q5 Q7 [III.ajk]This article does not aspire to empiricity.

1501. Lukas Rytz, Martin Odersky & Philipp Haller (2012): Lightweight Polymorphic Effects. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 258-282. doi:10.1007/978-3-642-31057-7_13 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1502. Didier Rémy & Jérôme Vouillon (1997): Objective ML: a simple object-oriented extension of ML. In Proc. 24th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 40-53. doi:10.1145/263699.263707 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1503. Bruno C. d. S. Oliveira (2009): Modular Visitor Components: A Practical Solution to the Expression Families Problem. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 269-293. doi:10.1007/978-3-642-03013-0_13 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1504. Pertti Saariluoma & Jorma Sajaniemi (1989): Visual information chunking in spreadsheet calculation. International Journal of Man-Machine Studies 30 (5). Pages 475-488. doi:10.1016/S0020-7373(89)80029-X Exclusion reasons: Q1–2 [II.ajk]This article reports a study on spreadsheet calculation,

which is (apart from the case of a single isolated cell) a nontextual language.
1505. H. Sackman, W. J. Erikson & E. E. Grant (1968): Exploratory experimental studies comparing online and offline programming performance. Communications of the ACM 11 (1). Pages 3-11. doi:10.1145/362851.362858 Exclusion reasons: Q1–2 [III.ajk]This article does not report a study evaluating programming language design decisions.
1506. Arun Saha (2011): Origins of poor code readability. In PPIG 2011. http://ppig.org/papers/23/31%20Saha.pdf Exclusion reasons: Q1–2 [II.ajk]No language design issue.
1507. S.K. Sahoo, J. Criswell & V. Adve (2010): An empirical study of reported bugs in server software with implications for automated bug diagnosis. Volume 1.In Software Engineering, 2010 ACM/IEEE 32nd International Conference on. Pages 485-494. doi:10.1145/1806799.1806870 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
1508. Chieri Saito & Atsushi Igarashi (2009): Self type constructors. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications. Pages 263-282. doi:10.1145/1640089.1640109 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article has no aspiration to empiricity.
1509. Jorma Sajaniemi (1999): Getting rid of the single notation paradigm with multiple views. In PPIG 1999. (Found in http://ppig.org/workshops/11th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.
1510. J. Sajaniemi (2002): An empirical analysis of roles of variables in novice-level procedural programs. In Human Centric Computing Languages and Environments, 2002. Proceedings. IEEE 2002 Symposia on. Pages 37-39. doi:10.1109/HCC.2002.1046340 Exclusion reasons: Q1–2 [II.ajk]Program analysis study; no PL design issue.
1511. Jorma Sajaniemi & Raquel Navarro Prieto (2005): Roles of Variables in Experts' Programming Knowledge. In PPIG 2005. (Found in http://ppig.org/workshops/17th-programme.html.) Exclusion reasons: Q5 [III.ajk]This article does not evaluate a language design decision.
1512. Jorma Sajaniemi & Marja Kuittinen (2005): An Experiment on Using Roles of Variables in Teaching Introductory Programming. Computer Science Education 15 (1). Pages 59-82. doi:10.1080/08993400500056563 Exclusion reasons: Q1–2 [II.ajk]Studies teaching approaches.
1513. Jorma Sajaniemi & Marja Kuittinen (2007): From Procedures to Objects: What Have We (Not) Done?. In PPIG 2007. http://www.ppig.org/papers/19th-Sajaniemi.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.
1514. Jorma Sajaniemi & Marja Kuittinen (2008): From procedures to objects: A research agenda for the psychology of object-oriented programming education. Human Technology 4 (1). Pages 75-91. https://jyx.jyu.fi/dspace/handle/123456789/20221 Exclusion reasons: Q7 [III.ajk]This article presents a nonsystematic map of empirical research in its topic and proposes further research topics. It does not discuss empirical evidence except conclusorily.
1515. Markku Sakkinen (1988): On the darker side of C++. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 162-176. doi:10.1007/3-540-45910-3_10 Exclusion reasons: Q5 [III.ajk]This analytical article does not aspire to empiricity.
1516. Lee Salzman & Jonathan Aldrich (2005): Prototypes with Multiple Dispatch: An Expressive and Dynamic Object Model. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 312-336. doi:10.1007/11531142_14 Exclusion reasons: Q1–2 [II.ajk]Theory development, no comparison in evaluation (based on the abstract).
1517. Hesam Samimi, Ei Darli Aung & Todd Millstein (2010): Falling Back on Executable Specifications. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 552-576. doi:10.1007/978-3-642-14107-2_26 Exclusion reasons: Q5 [III.ajk]This article contains "case studies" which are empirical only so far as implementation evaluation goes; language evaluation is quite analytic.
1518. Jean E. Sammet (1962): Basic elements of COBOL 61. Communications of the ACM 5 (5). Pages 237-253. doi:10.1145/367710.367721 Exclusion reasons: Q1–2 [III.ajk]This language tutorial does not report a study.
1519. Jean E. Sammet (1966): The use of English as a programming language. Communications of the ACM 9 (3). Pages 228-230. doi:10.1145/365230.365274 Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.
1520. Jean E. Sammet (1972): Programming languages: history and future. Communications of the ACM 15 (7). Pages 601-610. doi:10.1145/361454.361485 Exclusion reasons: Q1–2 [II.ajk]History work.
1521. Jean E. Sammet (1986): Why Ada is not just another programming language. Communications of the ACM 29 (8). Pages 722-732. doi:10.1145/6424.6425 Exclusion reasons: Q5 [II.ajk]Advocation piece; no study reported, based on the abstract.
1522. David Sandberg (1982): Lithe: a language combining a flexible syntax and classes. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 142-145. doi:10.1145/582153.582169 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1523. Bo Sanden (1985): Systems programming with JSP: example - a VDU controller. Communications of the ACM 28 (10). Pages 1059-1067. doi:10.1145/4372.4376 Exclusion reasons: Q1–2 [III.ajk]No comparison.
1524. Donald Sannella & Andrzej Tarlecki (1985): Program specification and development in standard ML. In Proc. 12th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 67-77. doi:10.1145/318593.318614 Exclusion reasons: Q5 [III.ajk]This analytical-construcive paper does not aspire to empiricity.
1525. Cláudio Nogueira Sant'Anna, Alessandro Fabricio Garcia, Christina von Flach Garcia Chavez, Carlos José Pereira de Lucena & Arndt von Staa (2003): On the Reuse and Maintenance of Aspect-Oriented Software: An Assessment Framework. Report. http://www.dbd.puc-rio.br/depto_informatica/03_26_santanna.pdf Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
1526. Vijay A. Saraswat & Martin Rinard (1990): Concurrent constraint programming. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 232-245. doi:10.1145/96709.96733 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1527. Vijay A. Saraswat, Radha Jagadeesan & Vineet Gupta (1995): Default timed concurrent constraint programming. In Proc. 22nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 272-285. doi:10.1145/199448.199513 Exclusion reasons: Q5 [II.ajk]Formal theoretical work.
1528. Vijay Saraswat & others (2006): Report on the experimental language X10. Technical report. http://domino.research.ibm.com/comm/research_projects.nsf/pages/x10.index.html/$FILE/ATTH4YZ5.pdf Exclusion reasons: Q1–2 [III.ajk]The URL is password encumbered now. However, all indications (including versions of the paper freely available, such as http://x10.sourceforge.net/documentation/languagespec/x10-174.pdf) point to this being a language exposition.
1529. V. Sartori, B. Mekuria Eshete & A. Villafiorita (2011): Measuring the Impact of Different Metrics on Software Quality: a Case Study in the Open Source Domain. In ICDS 2011, The Fifth International Conference on Digital Society. Pages 172-177. http://www.thinkmind.org/index.php?view=article&articleid=icds_2011_7_10_10100 Exclusion reasons: Q1–2 [III.ajk]Although this paper does report correlations between programming languages and some quality metrics, it doesn't in any meaningful sense evaluate the efficacy of any design decisions involved in the languages.
1530. Yoshiki Sato & Shigeru Chiba (2005): Loosely-Separated "Sister" Namespaces in Java. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 49-70. doi:10.1007/11531142_3 Exclusion reasons: Q1–2 [II.ajk]Language feature exposition and feasibility.
1531. M. Satpathy, N.T. Siebel & D. Rodriguez (2004): Assertions in object oriented software maintenance: analysis and case study. In Software Maintenance, 2004. Proceedings. 20th IEEE International Conference on. Pages 124-133. doi:10.1109/ICSM.2004.1357797 Exclusion reasons: Q1–2 [II.ajk]Programming patterns study, no PL design issue.
1532. Edwin H. Satterthwaite (1969): MUTANT 0.5: an experimental programming language. CS-TR-69-120 at Stanford University, Department of Computer Science. ftp://reports.stanford.edu/www/TR/CS-TR-69-120.html Exclusion reasons: Q5 [III.ajk]This technical report is mostly a language exposition and has no aspiration to empiricial evaluation.
1533. Craig Schaffert, Topher Cooper, Bruce Bullis & Mike Kilian Carrie Wilpolt (1986): An introduction to Trellis/Owl. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 9-16 . doi:10.1145/960112.28699 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1534. Benson H. Scheff (1966): A simple user-oriented compiler source language for programming automatic test equipment. Communications of the ACM 9 (4). Pages 258-266. doi:10.1145/365278.365297 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1535. S. Schlesinger & L. Sashkin (1967): POSE: a language for posing problems to a computer. Communications of the ACM 10 (5). Pages 279-285. doi:10.1145/363282.363298 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1536. Franz Schmalhofer, Ralph Bergmann, Stefan Boschert & Jörg Thoben (1993): Chapter 10 Learning Program Abstractions: Model and Empirical Validation. Volume 101.In Gerhard Strube and Karl F. Wender (ed.) The Cognitive Psychology of Knowledge.North-Holland. Advances in Psychology. Pages 203-231. doi:10.1016/S0166-4115(08)62659-X http://www.sciencedirect.com/science/article/pii/S016641150862659X Exclusion reasons: Q1–2 [II.ajk]No PL design issue.
1537. M. L. Schneider, K. Hirsh-Pasek & S. Nudelman (1984): An experimental evaluation of delimiters in a command language syntax. International Journal of Man-Machine Studies 20 (6). Pages 521-535. doi:10.1016/S0020-7373(84)80027-9 Exclusion reasons: Q1–2 [III.ajk]The syntactic choices investigated are so far removed from issues confronted by programming (as opposed to command) language designers that this study is unlikely to offer any

relevant insight to language design choice efficacy. Also, the efficacy may be a bit irrelevant for our use.

1538. W. Schupp (1976): BASEX: A programming language for process automation with minicomputers. Development and experience. at Freiburg Univ.(Germany). Inst. fuer Physik. http://www.ntis.gov/search/product.aspx?ABBR=N7724806 Exclusion reasons: Q4 [III.ajk]It looks like this report is in German. See http://archive.org/stream/directo00unit/directo00unit_djvu.txt
1539. J. T. Schwartz (1975): Automatic data structure choice in a language of very high level. Communications of the ACM 18 (12). Pages 722-728. doi: 10.1145/361227.361235 Exclusion reasons: Q1–2 [II.ajk]Implementation technique.
1540. Jan Schäfer & Arnd Poetzsch-Heffter (2010): JCoBox: Generalizing Active Objects to Concurrent Components. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 275-299. doi:10.1007/978-3-642-14107-2_13 Exclusion reasons: Q1–2 [II.ajk]Theoretical and implementation issues.
1541. Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz & Andrew P. Black (2003): Traits: Composable Units of Behaviour. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2743. Pages 248-274. doi:10.1007/978-3-540-45070-2_12 Exclusion reasons: Q5 [III.ajk]This article contains an analytical evaluation of the proposed language feature in the form of using them in a single application, to demonstrate the feature's applicability.
1542. Nathanael Schärli, Andrew P. Black & Stéphane Ducasse (2004): Object-oriented encapsulation for dynamically typed languages. In OOPSLA '04: Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 130-149. doi:10.1145/1028976.1028988 Exclusion reasons: Q5 [III.ajk]This article does not report any nontrivial empirical evaluations (for efficacy) of the design decisions presented.
1543. Nathanael Schärli, Stéphane Ducasse, Oscar Nierstrasz & Roel Wuyts (2004): Composable Encapsulation Policies. In Proc. ECOOP 2003 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 26-50. doi:10.1007/978-3-540-24851-4_2 Exclusion reasons: Q5 [III.ajk]The evaluation in this article is analytical, not empirical.
1544. Dana S. Scott (1977): Logic and programming languages. Communications of the ACM 20 (9). Pages 634-641. doi:10.1145/359810.359826 Exclusion reasons: Q1–2 [II.ajk]Formal theory building (developing conceptual understanding)
1545. David Scott, Richard Sharp, Thomas Gazagnaire & Anil Madhavapeddy (2010): Using functional programming within an industrial product group: perspectives and perceptions. In Proceedings of the 15th ACM SIGPLAN international conference on Functional programming. New York, NY, USA: ACM. ICFP '10. Pages 87-92. doi:10.1145/1863543.1863557 Exclusion reasons: Q1–2 [III.ajk]This article does not deal with any language design decisions.
1546. Marc M. Sebrechts & Paul H. Gross (1985): Programming in natural language: A descriptive analysis. Behavior Research Methods 17 (2). Pages 268-274. doi:10.3758/BF03214395 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions (unless the decision is to allow natural language use – but that's a bit too vague for my purposes, since an admissible language must be formal).
1547. João Costa Seco & Luís Caires (2000): A Basic Model of Typed Components. In Proc. ECOOP 2000 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1850. Pages 108-128. doi:10.1007/3-540-45102-1_6 Exclusion reasons: Q5 [II.ajk]Theoretical work.
1548. Judith Segal (1993): Empirical Studies of Learners of Functional Programming. In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]Studies learning, no language evaluation.
1549. Judith Segal (1994): Empirical studies of functional programming learners evaluating recursive functions. Instructional Science 22. Pages 385-411. doi:10.1007/BF00891962 Exclusion reasons: Q1–2 [III.ajk]This article studies learning outcomes, and has no language design relevance.
1550. Ed Seidewitz (1987): Object-oriented programming in Smalltalk and ADA. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 202-213. doi:10.1145/38765.38826 Exclusion reasons: Q5 [III.ajk]This article presents an analytical comparison of two languages; there is no aspiration for empiricity.
1551. Manuel Serrano (1999): Wide Classes. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 391-415. doi:10.1007/3-540-48743-3_18 Exclusion reasons: Q1–2 [II.ajk]Language exposition and implementation.
1552. Ravi Sethi (1980): A case study in specifying the semantics of a programming language. In Proceedings of the 7th ACM SIGPLAN-SIGACT symposium on Principles of programming languages. New York, NY, USA: ACM. Pages 117-130. doi:10.1145/567446.567458 Exclusion reasons: Q1–2 [II.ajk]Study of language description; no PL design issue.
1553. Ohad Shacham, Nathan Bronson, Alex Aiken, Mooly Sagiv, Martin Vechev & Eran Yahav (2011): Testing atomicity of composed concurrent operations. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 51-64. doi:10.1145/2048066.2048073 Exclusion reasons: Q1–2 [II.ajk]Testing technique, no language design issue.
1554. Russell L. Shackelford & Albert N. Badre (1993): Why can't smart students solve simple programming problems?. International Journal of Man-Machine Studies 38 (6). Pages 985-997. doi:10.1006/imms.1993.1045 Exclusion reasons: Q1–2 [III.ajk]This article studies teaching, not any language design decisions.
1555. H. Shahriar (1988): A Case Study on TCL Language. Student project report. http://research.cs.queensu.ca/~cordy/cisc860/Proejcts/TCL-Project-Shahriar.pdf Exclusion reasons: Q5 [III.ajk]This is a student analysis of TCL; it does not aspire to empiricity.
1556. Hao Shangfu, Zhang Xiao & Sun Baili (2009): The Virtual Experiment Design of Serial Communication Based on VC++. Volume 1.In Computer Science-Technology and Applications, 2009. IFCSTA '09. International Forum on. Pages 347-349. doi:10.1109/IFCSTA.2009.91 Exclusion reasons: Q1–2 [II.ajk]No programming language evaluation, based on the abstract.
1557. Stan Shannon & Claudia Henschke (1967): Stat-Pack: a biostatistical programming package. Communications of the ACM 10 (2). Pages 123-125. doi:10.1145/363067.363122 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
1558. Ehud Shapiro (1984): Systems programming in concurrent prolog. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 93-105. doi:10.1145/800017.800520 Exclusion reasons: Q1–2 [II.ajk]No comparison.
1559. Micha Sharir (1982): Some Observations Concerning Formal Differentiation of Set Theoretic Expressions. ACM Transactions on Programming Languages and Systems 4 (2). Pages 196-225. doi:10.1145/357162.357166 Exclusion reasons: Q1–2 [III.ajk]This paper has no language design relevance.
1560. Christopher J. Shaw (1963): A specification of JOVIAL. Communications of the ACM 6 (12). Pages 721-736. doi:10.1145/763973.763978 Exclusion reasons: Q1–2 [III.ajk]This is a language exposition, not a study report.
1561. Christopher J. Shaw (1964): On declaring arbitrarily coded alphabets. Communications of the ACM 7 (5). Pages 288-290. doi:10.1145/364099.364236 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1562. Mary Shaw (1974): Reduction of compilation costs through language contraction. Communications of the ACM 17 (5). Pages 245-250. doi:10.1145/360980.360989 Exclusion reasons: Q1–2 [III.ajk]This article reports a study in which subsets of Algol are created and the effects of each subsetting step to program cost is estimated. As such, it studies the general behaviour of language subsetting and not particular design decisions.
1563. Mary Shaw, William A. Wulf & Ralph L. London (1977): Abstraction and verification in Alphard: defining and specifying iteration and generators. Communications of the ACM 20 (8). Pages 553-564. doi:10.1145/359763.359782 Exclusion reasons: Q5 [II.ajk]Exposition and formal development of a language feature.
1564. I. Shcherbakov, C. Weis & N. Wehn (2011): Bringing C++ productivity to VHDL world: From language definition to a case study. In Specification and Design Languages (FDL), 2011 Forum on. Pages 1-7. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6069473 Exclusion reasons: Q1–2 [III.ajk]HDLs are not programming languages.
1565. Zhiyu Shen, Zhiyuan Li & Pen-Cchung Yew (1990): An empirical study of Fortran programs for parallelizing compilers . Parallel and Distributed Systems, IEEE Transactions on 1 (3). Pages 356-364. doi:10.1109/71.80162 Exclusion reasons: Q1–2 [II.ajk]This article studies programming language usage patterns for use in compiler development, and does not evaluate any language design decisions.
1566. S.B. Sheppard, B. Curtis, P. Milliman & T. Love (1979): Modern Coding Practices and Programmer Performance. Computer 12 (12). Pages 41-49. doi:10.1109/MC.1979.1658575 Exclusion reasons: Q1–2 [III.ajk]This article compares programming styles with no clear language design decisions at play.
1567. Peter B. Sheridan (1959): The arithmetic translator-compiler of the IBM FORTRAN automatic coding system. Communications of the ACM 2 (2). Pages 9-21. doi:10.1145/368280.368289 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1568. Mark Sherman (1984): Paragon: Novel uses of type hierarchies for data abstraction. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 208-217. doi:10.1145/800017.800532 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1569. Mark Shields & Erik Meijer (2001): Type-indexed rows. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 261-275. doi:10.1145/360204.360230 Exclusion reasons: Q1–2 [II.ajk]Type-theoretical development.
1570. Ben Shneidennan (1986): Empirical studies of programmers: The territory, paths, and destinations. In E. Soloway and R. Iyengar (ed.) Empirical Studies of Programmers. Norwood, NJ: Ablex. Pages 1-12. http://hcil2.cs.umd.edu/trs/86-02/86-02.txt Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate nor summarise studies evaluating programming language design decisions.

1571. Ben Shneiderman & Don McKay (1976): Experimental Investigations of Computer Program Debugging and Modification. In Proceedings of the Human Factors and Ergonomics Society Annual Meeting. Pages 557-563. doi:10.1177/154193127602002401 http://pro.sagepub.com/content/20/24/557.abstract Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.
1572. Ben Shneiderman, Richard Mayer, Don McKay & Peter Heller (1977): Experimental investigations of the utility of detailed flowcharts in programming. Communications of the ACM 20 (6). Pages 373-381. doi:10.1145/359605.359610 Exclusion reasons: Q1–2 [II.ajk]An interesting empirical study, but a flowcharting system is not a programming language according to the protocol's definition (not a set of strings).
1573. B. Shneiderman (1977): Measuring computer program quality and comprehension. International Journal of Man-Machine Studies 9 (4). Pages 465-478. doi:10.1016/S0020-7373(77)80014-X Exclusion reasons: Q1–2 [II.ajk]This article proposes and evaluates a method of measuring program comprehension.
1574. Ben Shneiderman (1980): Natural vs. precise concise languages for human operation of computers: research issues and experimental approaches. In Proceedings of the 18th annual meeting on Association for Computational Linguistics. Pages 139–141. doi:10.3115/981436.981478 Exclusion reasons: Q1–2 [III.ajk]This article is a persuasive piece about natural language user interfaces. It has references that should probably be snowballed but it otherwise clearly outside the scope of this study.
1575. Miriam G. Shoffner & Peter J. Brown (1963): A suggested method of making fuller use of strings in ALGOL 60. Communications of the ACM 6 (4). Pages 169-171. doi:10.1145/366349.366543 Exclusion reasons: Q5 [III.ajk]This article is constructive-analytical with no aspiration to empiricity.
1576. K. Shrestha & M. J. Rutherford (2011): An Empirical Evaluation of Assertions as Oracles. In Software Testing, Verification and Validation (ICST), 2011 IEEE Fourth International Conference on. Pages 110-119. doi:10.1109/ICST.2011.50 Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study in which existing JML-annotated Java programs were tested to see if the annotations could detect bugs. It does not evaluate any language design decisions.
1577. Nan C. Shu, Barron C. Housel & Vincent Y. Lum (1975): CONVERT: a high level translation definition language for data conversion. Communications of the ACM 18 (10). Pages 557-567 . doi:10.1145/361020.361023 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1578. Edgar H. Sibley & Robert W. Taylor (1973): A data definition and mapping language. Communications of the ACM 16 (12). Pages 750-759. doi: 10.1145/362552.362555 Exclusion reasons: Q5 Q7 [III.ajk]This article has no aspiration to empiricity.
1579. J. I. A Siddiqi (1984): An empirical investigation into the program design process. Interfaces in Computing 2 (3). Pages 279-293. doi:10.1016/0252-7308(84)90048-5 Exclusion reasons: Q1–2 [II.ajk]This article discusses program design. It does not evaluate any language design decisions.
1580. J. I. A. Siddiqi & B. Ratcliff (1989): Specification influences in program design. International Journal of Man-Machine Studies 31 (4). Pages 393-404. doi:10.1016/0020-7373(89)90002-3 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1581. Jawed Siddiqi & Babak Khazaei (1990): What are the 'Carry Over Effects' in changing from a Procedural to a Declarative Approach?. In PPIG 1990. (Found in http://ppig.org/workshops/2nd-programme.html.) Exclusion reasons: Q3 [III.ajk]This article does not appear to be available. An article with the same authors and title is at http://dx.doi.org/10.1109/CMPSAC.1989.65169, published a couple of months earlier; it does not report a language design decision evaluation study.
1582. Milton Siegel & Albert E. Smith (1962): Interim report on bureau of ships COBOL evaluation program. Communications of the ACM 5 (5). Pages 256-259. doi:10.1145/367710.367734 Exclusion reasons: Q1–2 [III.ajk]This article reports on a study evaluating COBOL with respect to the requirements of a particular user organization, but it does not evaluate the efficacy of particular design decisions.
1583. Jeremy Siek (2012): The C++0x "Concepts" Effort. Volume 7470.In Gibbons, Jeremy (ed.) Generic and Indexed Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 175-216. doi:10.1007/978-3-642-32202-0_4 Exclusion reasons: Q1–2 [II.ajk]This article does not report an evaluative study.
1584. M. E. Sime, A. T. Arblaster & T. R. G. Green (1977): Reducing programming errors in nested conditionals by prescribing a writing procedure. International Journal of Man-Machine Studies 9 (1). Pages 119-126. doi:10.1016/S0020-7373(77)80046-1 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1585. Christopher Simpkins, Sooraj Bhat, Jr. Charles Isbell & Michael Mateas (2008): Towards adaptive programming: integrating reinforcement learning into a programming language. In OOPSLA '08: Proceedings of the 23rd ACM SIGPLAN conference on Object-oriented programming systems languages and applications. Pages 603-614. doi:10.1145/1449764.1449811 Exclusion reasons: Q1–2 [III.ajk]This article does not report an evaluative empirical study.
1586. Renuka Sindhgatta, Bikram Sengupta & Subhajit Datta (2011): Coping with distance: an empirical study of communication on the jazz platform. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 155-162. doi:10.1145/2048147.2048190 Exclusion reasons: Q1–2 [II.ajk]No language design issue.
1587. Frank Singhoff & Alain Plantec (2007): Towards User-Level Extensibility of an Ada Library: An Experiment with Cheddar. Volume 4498.In Abdennahder, Nabil and Kordon, Fabrice (ed.) Reliable Software Technologies – Ada Europe 2007. Lecture Notes in Computer Science. Pages 180-191. doi:10.1007/978-3-540-73230-3_14 Exclusion reasons: Q1–2 [II.ajk]No programming language design relevance.
1588. A. P. Sinha & I. Vessey (1992): Cognitive fit: an empirical study of recursion and iteration. Software Engineering, IEEE Transactions on 18 (5). Pages 368-379. doi:10.1109/32.135770 Exclusion reasons: Q1–2 [III.ajk]This article reports a laboratory experiment assessing the theory of cognitive fit with respect to recursion and iteration in the context of both LISP and PASCAL. While the results have some relevance to language design, the study does not actually evaluate any language design decisions.
1589. Jeffrey Mark Siskind & Barak A. Pearlmutter (2007): First-class nonstandard interpretations by opening closures. In Proc. 34th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 71-76. doi:10.1145/1190216.1190230 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1590. D.I.K. Sjøberg, J.E. Hannay, O. Hansen, V.B. Kampenes, A. Karahasanovic, N.-K. Liborg & A.C. Rekdal (2005): A survey of controlled experiments in software engineering. Software Engineering, IEEE Transactions on 31 (9). Pages 733-753. doi:10.1109/TSE.2005.97 Exclusion reasons: Q1–2 [III.ajk]This article does not summarise or consolidate research.
1591. Therapon Skotiniotis, Jeffrey Palm & Karl Lieberherr (2006): Demeter Interfaces: Adaptive Programming Without Surprises. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 477-500. doi:10.1007/11785477_27 Exclusion reasons: Q5 [III.ajk]This constructive-analytical paper does not aspire to empiricity.
1592. A.R.A. Skyrme, N.L.R. Rodriguez, P.M. Musa, R. Ierusalimschy & B.O. Silvestre (2011): Embedding Concurrency: A Lua Case Study. Technical report. ftp://ftp.inf.puc-rio.br/pub/docs/techreports/11_13_skyrme.pdf Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate the efficacy of the construct, only its performance.
1593. D. SLIPPER & A.A. McEWAN (2011): A Systems Re-engineering Case Study: Programming Robots with occam and Handel-C. Communicating Process Architectures 2011: WoTUG-33 68. Pages 317. http://wotug.kent.ac.uk/papers/CPA-2011/SlipperMcEwan11/SlipperMcEwan11.pdf Exclusion reasons: Q1–2 [III.ajk]This article investigates reengineering an implementation component and does not in any real sense evaluate any language design decisions.
1594. Lee G. Slocum (1988): Device driver development in Ada: a case study. In Proceedings of the fifth Washington Ada symposium on Ada. New York, NY, USA: ACM. WADAS '88. Pages 25–33. doi:10.1145/339938.339954 Exclusion reasons: Q1–2 [III.ajk]This article discusses a use case for Ada; there is no PL design issue.
1595. Paul Slonaker, Mark S. Smith, Sharon Prizant & Judith M. Giles (1987): Development of multi-tasking software in Ada \— a case study. In Proceedings of the Joint Ada conference fifth national conference on Ada technology and fourth Washington Ada Symposium. Washington, DC, USA: George Washington University. Pages 304-317. http://dl.acm.org/citation.cfm?id=339771.339934 Exclusion reasons: Q3 [III.ajk]Interlibrary loan service was not able to locate an available copy. See msgid <920E91CF45A8354B9375277840FAEEC70C23D891@mbs2.ad.jyu.fi> in misc/interlibrary-service-rejects.mbox
1596. Luit J. Slooten, Fransisco Batle & Jesus Carrera (2011): An experimental approach to the performance penalty of the use of classes in Fortran 95. Advances in Engineering Software 42 (10). Pages 735-742. doi:10.1016/j.advengsoft.2011.05.011 Exclusion reasons: Q1–2 [III.ajk]This article studies implementation efficiency, and is not relevant to this study.
1597. Yannis Smaragdakis & Don Batory (1998): Implementing layered designs with mixin layers. In Proc. ECOOP'98 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1445. Pages 550-570. doi:10.1007/BFb0054107 Exclusion reasons: Q1–2 [III.ajk]This article describes a programming technique and presents no PL design issue.
1598. Yannis Smaragdakis, Anthony Kay, Reimer Behrends & Michal Young (2007): Transactions with isolation and cooperation. In OOPSLA '07: Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications. Pages 191-210. doi:10.1145/1297027.1297042 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1599. Yannis Smaragdakis, Jacob Evans, Caitlin Sadowski, Jaeheon Yi & Cormac Flanagan (2012): Sound predictive race detection in polynomial time. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages. Pages 387-400. doi:10.1145/2103656.2103702 Exclusion reasons: Q1–2 [II.ajk]This article deals with a program trace analysis technique.

1600. Joseph W. Smith (1960): Syntactic and semantic augments to ALGOL. Communications of the ACM 3 (4). Pages 211-213. doi:10.1145/367177.367210 Exclusion reasons: Q5 [III.ajk]This short language extension exposition does not aspire to empiricity.

1601. Brian Cantwell Smith (1984): Reflection and semantics in LISP. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 23-35. doi:10.1145/800017.800513 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1602. D. Douglas Smith (1988): Alexi — a case study in design issues for LISP capabilities in Ada. In Proceedings of the fifth Washington Ada symposium on Ada. New York, NY, USA: ACM. Pages 109-116. doi:10.1145/339938.339978 Exclusion reasons: Q5 [III.ajk]To the extent that this article evaluates the language, it does it analytically, by exploring the implications of the design, without aspiration to empiricity.

1603. Gordon N. Smith, Wesley M. Cohen, William E. Hefley & Daniel A. Levinthal (1989): Understanding the Adoption of Ada: A Field Study Report. CMU/SEI-89-TR-028 ESD-TR-88-037 at Carnegie Mellon University Software Engineering Institute. http://www.sei.cmu.edu/library/abstracts/reports/89tr028.cfm Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1604. Randall B. Smith (1994): Prototype-based languages (panel): object lessons from class-free programming. In OOPSLA '94: Proceedings of the ninth annual conference on Object-oriented programming systems, language, and applications. Pages 102-112. doi:10.1145/191080.191101 Exclusion reasons: Q5 [III.ajk]This abstract collection does not aspire to empiricity.

1605. Randall B. Smith & David Ungar (1995): Programming as an Experience: The Inspiration for Self. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 303-330. doi:10.1007/3-540-49538-X_15 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1606. Walter R. Smith (1995): Using a prototype-based language for user interface: the Newton project's experience. In OOPSLA '95: Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications. Pages 61-72. doi:10.1145/217838.217844 Exclusion reasons: Q5 [III.ajk]This is an experience report and as such excluded under our protocol.

1607. Geoffrey Smith & Dennis Volpano (1998): Secure information flow in a multi-threaded imperative language. In Proc. 25th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 355-364. doi:10.1145/268946.268975 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1608. Charles Smith & Sophia Drossopoulou (2005): Chai: Traits for Java-Like Languages. In Proc. ECOOP 2005 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3586. Pages 453-478. doi:10.1007/11531142_20 Exclusion reasons: Q5 [II.ajk]Formal theoretical study.

1609. B. H. Smith & L. Williams (2007): An Empirical Evaluation of the MuJava Mutation Operators. In Testing: Academic and Industrial Conference Practice and Research Techniques - MUTATION, 2007. TAICPART-MUTATION 2007. Pages 193-202. doi:10.1109/TAIC.PART.2007.12 http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4344124 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1610. Jon Sneyers, Tom Schrijvers & Bart Demoen (2009): The computational power and complexity of constraint handling rules. ACM Transactions on Programming Languages and Systems 31 (2). doi:10.1145/1462166.1462169 Exclusion reasons: Q1–2 [II.ajk]Theoretical development.

1611. C. F. Snook & R. Harrison (2004): Experimental comparison of the comprehensibility of a Z specification and its implementation in Java. Information and Software Technology 46 (14). Pages 955-971. doi:10.1016/j.infsof.2004.04.003 Exclusion reasons: Q1–2 [II.ajk]Compares specification and implementation, not two implementation notations.

1612. Alan Snyder (1986): Encapsulation and inheritance in object-oriented programming languages. In Conference proceedings on Object-oriented programming systems, languages and applications (OOPSLA '86). Pages 38-45. doi:10.1145/28697.28702 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1613. Stefan Sobernig, Patrick Gaubatz, Mark Strembeck & Uwe Zdun (2011): Comparing complexity of API designs: an exploratory experiment on DSL-based framework integration. In Proceedings of the 10th ACM international conference on Generative programming and component engineering. New York, NY, USA: ACM. Pages 157-166. doi:10.1145/2047862.2047890 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1614. Yuriy Solodkyy, Gabriel Dos Reis & Bjarne Stroustrup (2012): Open and efficient type switch for C++. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 963-982. doi:10.1145/2384616.2384686 Exclusion reasons: Q1–2 [III.ajk]The empirical evaluation focuses on performance providing no efficacy insights.

1615. Marvin Solomon (1975): Modes, values and expressions. In Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 149-159. doi:10.1145/512976.512992 Exclusion reasons: Q5 [III.ajk]This article presents a conceptual analysis, with no empirical aspirations.

1616. Elliot Soloway & Kate Ehrlich (1984): Empirical Studies of Programming Knowledge. IEEE Transactions on Software Engineering 10 (5). Pages 595-609. doi:10.1109/TSE.1984.5010283 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1617. Jon A. Solworth (1992): Epochs. ACM Transactions on Programming Languages and Systems 14 (1). Pages 28-53. doi:10.1145/111186.116785 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1618. Sooel Son, Kathryn S. McKinley & Vitaly Shmatikov (2011): RoleCast: finding missing security checks when you do not know what checks are. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 1069-1084. doi:10.1145/2048066.2048146 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design issues.

1619. Cleidson R. B. de Souza, David Redmiles, Li-Te Cheng, David Millen & John Patterson (2004): Sometimes you need to see through walls: a field study of application programming interfaces. In Proceedings of the 2004 ACM conference on Computer supported cooperative work. New York, NY, USA: ACM. Pages 63-71. doi:10.1145/1031607.1031620 Exclusion reasons: Q1–2 [II.ajk]An empirical study on an SE practice, not directly relevant to the question of what would be the best language mechanism for supporting that practice.

1620. Diomidis Spinellis, Vassilios Karakoidas & Panos Louridas (2012): Comparative language fuzz testing: programming languages vs. fat fingers. In Proceedings of the ACM 4th annual workshop on Evaluation and usability of programming languages and tools. New York, NY, USA: ACM. PLATEAU '12. Pages 25-34. doi:10.1145/2414721.2414727 Exclusion reasons: Q1–2 [III.ajk]This article compares a number of languages (in particular implementations) in how they cope with erroneous program mutations. There is no clear design decision nor efficacy facet at play.

1621. James C. Spohrer & Elliot Soloway (1986): Novice mistakes: are the folk wisdoms correct?. Communications of the ACM 29 (7). Pages 624-632. doi:10.1145/6138.6145 Exclusion reasons: Q1–2 [III.ajk]This empirical article does not evaluate a language design decision.

1622. C. R. Spooner (1986): The ML approach to the readable all-purpose language. ACM Transactions on Programming Languages and Systems 8 (2). Pages 215-243. doi:10.1145/5397.5918 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1623. Wilfred Springer (2009): Bit syntax for Java. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 609-612. doi:10.1145/1639950.1639955 Exclusion reasons: Q1–2 [III.ajk]This article describes a system: there is no PL design issue evident.

1624. Manu Sridharan, Shay Artzi, Marco Pistoia, Salvatore Guarnieri, Omer Tripp & Ryan Berg (2011): F4F: taint analysis of framework-based web applications. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. OOPSLA '11. Pages 1053-1068. doi:10.1145/2048066.2048145 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1625. Sriram Srinivasan & Alan Mycroft (2008): Kilim: Isolation-Typed Actors for Java (A Million Actors, Safe Zero-Copy Communication). In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 104-128. doi:10.1007/978-3-540-70592-5_6 Exclusion reasons: Q5 [III.ajk]This article includes a performance evaluation based on undisclosed-code microbenchmarks chosen for their analytical relevance instead of real-world prevalence or representativeness. Thus, it is unlikely it can be considered an empirical evaluation of the language design presented.

1626. Amitabh Srivastava (1992): Unreachable procedures in object-oriented programming. ACM Transactions on Programming Languages and Systems 1 (4). Pages 355-364. doi:10.1145/161494.161517 Exclusion reasons: Q1–2 [III.ajk]This article studies the prevalence of dead-code procedures in real-life programs written in various languages. It highlights the importance of interprocedural dead code elimination, which is an implementation concern, but does not evaluate language design decisions.

1627. Vincent St-Amour, Sam Tobin-Hochstadt, Matthew Flatt & Matthias Felleisen (2010): Where are you going with those types?. In Implementation and Application of Functional Languages, 22nd International Symposium (IFL 2010). (Presented at the IFL 2010 conference but does not appear in the formal proceedings.) http://www.ccs.northeastern.edu/home/stamourv/papers/tr-opt.pdf Exclusion reasons: Q1–2 [II.ajk]This article discusses an implementation technique, not a languagr design issue.

1628. Vincent St-Amour, Sam Tobin-Hochstadt & Matthias Felleisen (2012): Optimization coaching: optimizers learn to communicate with programmers. In Proceedings of the ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 163-178. doi:10.1145/2384616.2384629 Exclusion reasons: Q1–2 [III.ajk]The construct under discussion is an interactive tool and thus not a programming language.

1629. Nenad Stankovic, Dieter Kranzlmüller & Kang Zhang (2001): The PCG: An Empirical Study. Journal of Visual Languages & Computing 12 (2). Pages 203-216. doi:10.1006/jvlc.2000.0189 Exclusion reasons: Q1–2 [II.ajk]A visual language is not a PL under our definition.
1630. Mike Stark (1993): Impacts of object-oriented technologies: seven years of SEL studies. In OOPSLA '93: Proceedings of the eighth annual conference on Object-oriented programming systems, languages, and applications. Pages 365-373. doi:10.1145/165854.165925 Exclusion reasons: Q1–2 [III.ajk]This article does not report not review studies evaluating language design decisions.
1631. Allan M. Stavely (1993): An empirical study of iteration in applications software. Journal of Systems and Software 22 (3). Pages 167-177. doi:10.1016/0164-1212(93)90108-A Exclusion reasons: Q1–2 [II.ajk]Programming pattern study.
1632. Guy L. Steele, Jr. (1990): Making asynchronous parallelism safe for the world. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 218-231. doi:10.1145/96709.96731 Exclusion reasons: Q5 [III.ajk]This is an analytical paper, and has no empirical aspirations.
1633. J. Steensgaard-Madsen (1981): A Statement-Oriented Approach to Data Abstraction. ACM Transactions on Programming Languages and Systems 3 (1). Pages 1-10. doi:10.1145/357121.357122 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1634. Andreas Stefik, Susanna Siebert, Kim Slattery & Melissa Stefik (2011): Toward Intuitive Programming Languages. In Proceedings of the 2011 IEEE 19th International Conference on Program Comprehension. Washington, DC, USA: IEEE Computer Society. ICPC '11. Pages 213-214. doi:10.1109/ICPC.2011.33 Exclusion reasons: Q1–2 [III.ajk]This article does not report a study.
1635. Andreas Stefik, Christopher Hundhausen & Robert Patterson (2011): An empirical investigation into the design of auditory cues to enhance computer program comprehension. International Journal of Human-Computer Studies 69 (12). Pages 820-838. doi:10.1016/j.ijhcs.2011.07.002 Exclusion reasons: Q1–2 [II.ajk]This article deals with the use of sounds, and is thus not about a programming language design issue as we have defined it.
1636. M. Steinberg (2011): What is the impact of static type systems on maintenance tasks? An empirical study of differences in debugging time using statically and dynamically typed languages. . Master's Thesis. Exclusion reasons: Q3 [III.ajk]Unpublished thesis.
1637. S. L. Stewart (1975): Staple, an experimental structured programming language. Computer Languages 1 (1). Pages 61-71. doi:10.1016/0096-0551(75)90008-9 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1638. Glenn F. Stewart (1975): An algebraic model for string patterns. In Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 167-184. doi:10.1145/512976.512994 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1639. Patrick Steyaert, Wim Codenie, Theo D'Hondt, Koen De Hondt, Carine Lucas & Marc Van Limberghen (1993): Nested Mixin-Methods in Agora. In Proc. ECOOP'93 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 707. Pages 197-219. doi:10.1007/3-540-47910-4_12 Exclusion reasons: Q5 [II.ajk]Theoretical study.
1640. Patrick Steyaert & Wolfgang De Meuter (1995): A Marriage of Class- and Object-Based Inheritance Without Unwanted Children. In Proc. ECOOP'95 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 952. Pages 127-144. doi:10.1007/3-540-49538-X_7 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical study.
1641. Dan N. Stone, Eleanor W. Jordan & M. Keith Wright (1990): The impact of Pascal education on debugging skill. International Journal of Man-Machine Studies 33 (1). Pages 81-95. doi:10.1016/S0020-7373(05)80116-6 Exclusion reasons: Q1–2 [II.ajk]Evaluates the efficacy of a language in teaching progrmming, which is not the same thing as efficacy in programming. In this case, the difference is clear and large.
1642. Roger Stone (2001): The Science of Web-Programming. In PPIG 2001. (Found in http://ppig.org/workshops/13th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate a language design decision.
1643. Margaret-Anne Storey (2006): Theories, tools and research methods in program comprehension: past, present and future. Software Quality Journal 14 (3). Pages 187-208. doi:10.1007/s11219-006-9216-4 Exclusion reasons: Q1–2 [III.ajk]This article does not deal with language design decisions.
1644. Margaret Anne Storey (2008): Source Code Comments: Graffiti or Information? . In PPIG 2008. (Found in http://ppig.org/workshops/20th-programme.html.) Exclusion reasons: Q3 [III.ajk]Selection decision based on page 8 of http://www.cs.st-andrews.ac.uk/~jr/papers/ppig08Proceedings.pdf. This is a keynote speech of which only an abstract has been, so far as I can tell, published.
1645. Sven Stork, Paulo Marques & Jonathan Aldrich (2009): Concurrency by default: using permissions to express dataflow in stateful programs. In OOPSLA '09: Proceeding of the 24th ACM SIGPLAN conference companion on Object oriented programming systems languages and applications. Pages 933-940. doi:10.1145/1639950.1640060 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1646. C. Strachey (1961): Bitwise operations. Communications of the ACM 4 (3). Pages 146. doi:10.1145/366199.366254 Exclusion reasons: Q1–2 [III.ajk]This article is a brief note on a programming technique; there is no PL design issue at hand.
1647. C. Strachey & M. V. Wilkes (1961): Some proposals for improving the efficiency of ALGOL 60. Communications of the ACM 4 (11). Pages 488-491. doi:10.1145/366813.366816 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1648. E. de Sturler (1998): Object Oriented Programming in High Performance Fortran. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 578. doi:10.1007/3-540-49255-0_154 Exclusion reasons: Q1–2 [III.ajk]This article does not appear to be reporting a study.
1649. Jeffrey Stylos, Steven Clarke & Brad Myers (2006): Comparing API Design Choices with Usability Studies: A Case Study and Future Directions. In PPIG 2006. (Found in http://ppig.org/workshops/18th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This does not appear to evaluate language design decisions.
1650. Jeffrey Stylos & Steven Clarke (2007): Usability Implications of Requiring Parameters in Objects' Constructors. In Proceedings of the 29th international conference on Software Engineering. Washington, DC, USA: IEEE Computer Society. ICSE '07. Pages 529-539. doi:10.1109/ICSE.2007.92 Exclusion reasons: Q1–2 [III.ajk]While API design issues sometimes mirror language design issues, this study does not seem to transfer.
1651. J. Stylos, B. Graf, D. K. Busse, C. Ziegler, R. Ehret & J. Karstens (2008): A case study of API redesign for improved usability. In Visual Languages and Human-Centric Computing, 2008. VL/HCC 2008. IEEE Symposium on. Pages 189-192. doi:10.1109/VLHCC.2008.4639083 Exclusion reasons: Q1–2 [II.ajk]No programming-language relevance.
1652. Vivy Suhendra, Abhik Roychoudhury & Tulika Mitra (2010): Scratchpad allocation for concurrent embedded software. ACM Transactions on Programming Languages and Systems 32 (4). doi:10.1145/1734206.1734210 Exclusion reasons: Q1–2 [II.ajk]Implementation technique at best.
1653. Joshua Sunshine (2010): Unsticking the web. In SPLASH '10 Systems Programming Languages and Applications: Software for Humanity. Pages 223-224. doi:10.1145/1869542.1869584 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1654. Joshua Sunshine, Karl Naden, Sven Stork, Jonathan Aldrich & Éric Tanter (2011): First-class state change in plaid. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 713-732. doi:10.1145/2048066.2048122 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1655. Norihisa Suzuki (1981): Inferring types in Smalltalk. In Proc. 8th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 187-199. doi:10.1145/567532.567553 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1656. Norihisa Suzuki (1982): Analysis of pointer "rotation". Communications of the ACM 25 (5). Pages 330-335. doi:10.1145/358506.358513 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1657. Daniel C. Swinehart, Polle T. Zellweger, Richard J. Beach & Robert B. Hagmann (1986): A structural view of the Cedar programming environment. ACM Transactions on Programming Languages and Systems 8 (4). Pages 215-243. doi:10.1145/6465.6466 Exclusion reasons: Q5 [III.ajk]This article is a system exposition with no aspiration to empiricity (despite the short sections styled "case studies").
1658. D. M. Symes (1975): New control structures to aid gotolessness. In Proc. 2nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 194-203. doi:10.1145/512976.512996 Exclusion reasons: Q5 [III.ajk]This analytic-constructive paper has no aspiration to empiricity.
1659. Duane Szafron & Jonathan Schaeffer (1996): An experiment to measure the usability of parallel programming systems. Concurrency: Practice and Experience 8 (2). Pages 147-166. doi:10.1002/(SICI)1096-9128(199603)8:2<147::AID-CPE199>3.0.CO;2-O Exclusion reasons: Q1–2 [III.ajk]This article compares a particular visual language to a textual one. There is no admissible language design question at play, since visual languages are specifically excluded.
1660. Martha R. Szczur & Philip Miller (1988): Transportable applications environment (TAE) plus experiences in "Object"-ively modernizing a user interface environment. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 58-70. doi:10.1145/62083.62090 Exclusion reasons: Q5 [III.ajk]This experience report does not aspire to empiricity.
1661. Clemens A. Szyperski (1992): Import is not inheritance why we need both: Modules and classes. In Proc. ECOOP'92 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 615. Pages 19-32. doi:10.1007/BFb0053028 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.
1662. Miklós Szőts (1984): A Comparison of two Logic Programming Languages: A Case Study. In Proceedings of the Second International Logic Programming Conference, Uppsala University, Uppsala, Sweden, July 2–6, 1984. Pages 41–51. Exclusion reasons: Q5 [III.ajk]This article presents an analytical

comparison of PROLOG and LOBO; it is not empirical.

1663. Fernando Sánchez, Juan Hernández, Juan Manuel Murillo & Enrique Pedraza (1998): Run-Time Adaptability of Synchronization Policies in Concurrent Object Oriented Languages. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 64. doi:10.1007/3-540-49255-0_140 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.

1664. S. Tucker Taft, Joshua Bloch, Robert Bocchino, Sebastian Burckhardt, Hassan Chafi, Russ Cox, Benedict Gaster, Guy Steele & David Ungar (2011): Multicore, manycore, and cloud computing: is a new programming language paradigm required?. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 165-170. doi:10.1145/2048147.2048192 Exclusion reasons: Q1–2 [II.ajk]Panel report, not a study report.

1665. Kazunori Takashio & Mario Tokoro (1992): DROL: an object-oriented programming language for distributed real-time systems. In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications. Pages 276-294. doi:10.1145/141936.141959 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1666. Samira Tasharofi, Peter Dinges & Ralph Johnson (2012): Why Do Scala Developers Mix the Actor Model with Other Concurrency Models?. Technical Report. http://hdl.handle.net/2142/34816 Exclusion reasons: Q1–2 [III.ajk]This article does not study efficacy, nor is there a clear design decision at play,

1667. Barbee E. Teasley (1994): The effects of naming style and expertise on program comprehension. International Journal of Human-Computer Studies 40 (5). Pages 757-770. doi:10.1006/ijhc.1994.1036 Exclusion reasons: Q1–2 [III.ajk]This article does not study any language design issue.

1668. DAVID P. TEGARDEN & STEVEN D. SHEETZ (2001): Cognitive activities in OO development. International Journal of Human-Computer Studies 54 (6). Pages 779-798. doi:10.1006/ijhc.1999.0462 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate language design decisions.

1669. Daniel Teichroew & John Francis Lubin (1966): Computer simulation - discussion of the technique and comparison of languages. Communications of the ACM 9 (10). Pages 723-741. doi:10.1145/365844.365851 Exclusion reasons: Q5 [III.ajk]This article is an analytical review of several simulation programming languages. It has no aspiration for empiricality.

1670. C. Teijeiro, G. L. Taboada, J. Touriño, B. B. Fraguela, R. Doallo, D. A. Mallón, A. Gómez, J. C. Mouriño & B. Wibecan (2009): Evaluation of UPC programmability using classroom studies. In Proceedings of the Third Conference on Partitioned Global Address Space Programing Models. New York, NY, USA: ACM. PGAS '09. Pages 10:1-10:7. doi:10.1145/1809961.1809975 Exclusion reasons: Q5 [III.ajk]This article does not, in any real sense, evaluate empirically language design decisions (lack of empirical comparison)

1671. Ewan Tempero, James Noble & Hayden Melton (2008): How Do Java Programs Use Inheritance? An Empirical Study of Inheritance in Java Software. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 667-691. doi:10.1007/978-3-540-70592-5_28 Exclusion reasons: Q5 [III.ajk]This article empirically describes actual usage patterns for inheritance in Java. It does not evaluate a language design decision in any relevant sense.

1672. E. Tempero (2009): How Fields are Used in Java: An Empirical Study. In Software Engineering Conference, 2009. ASWEC '09. Australian. Pages 91-100. doi:10.1109/ASWEC.2009.19 Exclusion reasons: Q1–2 [II.ajk]Programming patterns study; no PL design issue.

1673. E. Tempero, C. Anslow, J. Dietrich, T. Han, Jing Li, M. Lumpe, H. Melton & J. Noble (2010): The Qualitas Corpus: A Curated Collection of Java Code for Empirical Studies. In Software Engineering Conference (APSEC), 2010 17th Asia Pacific. Pages 336-345. doi:10.1109/APSEC.2010.46 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1674. Tachio Terauchi & Alex Aiken (2008): Witnessing side effects. ACM Transactions on Programming Languages and Systems 30 (3). doi:10.1145/1353445.1353449 Exclusion reasons: Q1–2 [II.ajk]Formal development only.

1675. J. W. Thatcher, E. G. Wagner & J. B. Wright (1982): Data Type Specification: Parameterization and the Power of Specification Techniques. ACM Transactions on Programming Languages and Systems 4 (4). Pages 711-732. doi:10.1145/69622.357192 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1676. Satish Thatte (1990): Quasi-static typing. In Proc. 17th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 367-381. doi:10.1145/96709.96747 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1677. Hayo Thielecke (2003): From control effects to typed continuation passing. In Proc. 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 139-149. doi:10.1145/604131.604144 Exclusion reasons: Q5 [III.ajk]This article reports theoretical work with no empirical aspirations.

1678. Kresten Krab Thorup (1997): Genericity in java with virtual types. In Proc. ECOOP'97 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1241. Pages 444-471. doi:10.1007/BFb0053390 Exclusion reasons: Q1–2 [II.ajk]No efficacy evaluation.

1679. Kresten Krab Thorup & Mads Torgersen (1999): Unifying Genericity: Combining the Benefits of Virtual Types and Parameterized Classes. In ECOOP'99 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 1628. Pages 186-204. doi:10.1007/3-540-48743-3_9 Exclusion reasons: Q1–2 [II.ajk]Language exposition

1680. Ferdian Thung, Lucia, David Lo, Lingxiao Jiang, Foyzur Rahman & Premkumar T. Devanbu (2012): To what extent could we detect field defects? an empirical study of false negatives in static bug finding tools. In Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering. New York, NY, USA: ACM. ASE 2012. Pages 50-59. doi:10.1145/2351676.2351685 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1681. Walter F. Tichy, Paul Lukowicz, Lutz Prechelt & Ernst A. Heinz (1995): Experimental evaluation in computer science: A quantitative study. Journal of Systems and Software 28 (1). Pages 9-18. doi:10.1016/0164-1212(94)00111-Y Exclusion reasons: Q1–2 [III.ajk]This article does not deal with any language design decisions.

1682. Ben L. Titzer (2006): Virgil: objects on the head of a pin. In OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. Pages 191-208. doi:10.1145/1167473.1167489 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1683. Robert G. Tobey (1966): Eliminating monotonous mathematics with FORMAC. Communications of the ACM 9 (10). Pages 742-751. doi:10.1145/365844.365857 Exclusion reasons: Q1–2 [II.ajk]Language exposition; no comparison.

1684. C. Tomlinson & V. Singh (1989): Inheritance and synchronization with enabled-sets. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA 89). Pages 103-112. doi:10.1145/74877.74889 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1685. Mads Torgersen (2004): The Expression Problem Revisited: Four New Solutions Using Generics. In Proc. ECOOP 2004 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 3086. Pages 123-146. doi:10.1007/978-3-540-24851-4_6 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1686. Noah Torp-Smith, Lars Birkedal & John C. Reynolds (2008): Local reasoning about a copying garbage collector. ACM Transactions on Programming Languages and Systems 30 (4). doi:10.1145/1377492.1377499 Exclusion reasons: Q1–2 [II.ajk]Formal development, no comparison.

1687. Weslley Torres, Gustavo Pinto, Benito Fernandes, João Paulo Oliveira, Filipe Alencar Ximenes & Fernando Castor (2011): Are Java programmers transitioning to multicore?: a large scale study of java FLOSS. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 123-128. doi:10.1145/2095050.2095072 Exclusion reasons: Q1–2 [III.ajk]This article reports a study determining whether Java concurrency is actually used in real programs. It does not evaluate their efficacy.

1688. Prabhat Totoo, Pantazis Deligiannis & Hans-Wolfgang Loidl (2012): Haskell vs. F# vs. Scala: a high-level language features and parallelism support comparison. In Proceedings of the 1st ACM SIGPLAN workshop on Functional high-performance computing. New York, NY, USA: ACM. FHPC '12. Pages 49-60. doi:10.1145/2364474.2364483 Exclusion reasons: Q5 [III.ajk]This article compares several languages for implementation efficiency and, analytically (or perhaps as an experience report), for programmability. Efficacy is not empirically considered.

1689. Jesse A. Tov & Riccardo Pucella (2011): Practical affine types. In Proc. 38th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 447-458. doi:10.1145/1926385.1926436 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1690. Laurence Tratt (2008): Domain specific language implementation via compile-time meta-programming. ACM Transactions on Programming Languages and Systems 30 (6). doi:10.1145/1391956.1391958 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1691. D. Trentesaux, P. Pesin & C. Tahon (2001): Comparison of constraint logic programming and distributed problem solving: a case study for interactive, efficient and practicable job-shop scheduling. Computers & Industrial Engineering 39 (1-2). Pages 187-211. doi:10.1016/S0360-8352(00)00078-4 http://www.sciencedirect.com/science/article/pii/S0360835200000784 Exclusion reasons: Q1–2 [III.ajk]This paper has no relevance to programming language design.

1692. J. M. Triance & J. F. S. Yow (1980): MCOBOL—a prototype macro facility for Cobol. Communications of the ACM 23 (8). Pages 432-439. doi:10.1145/358896.358889 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1693. Howard Trickey (1988): C++ versus LISP: a case study. SIGPLAN Notices 23 (2). Pages 9-18. doi:10.1145/43908.43909 Exclusion reasons: Q5 [III.ajk]This article reports on a study in which its author wrote the same program in two languages and used the resulting programs to compare the languages. As

such, it can be taken as evaluating the efficacy of choosing to design a C++-like language instead of a Lisp-like language. The quality of the study can be assessed based on this report. Quality assessment is not a part of this selection decision process, but this study has so many methodological flaws that it cannot be said to be presenting empirical evidence, by our definition, on its claims. A fatal flaw is that the two programs were not independently programmed, and in fact they were developed together, including the occasional transliteration of code from one language to the other.

1694. Emma Triffitt & Babak Khazaei (2002): A Study of Usability of Z Formalism Based on Cognitive Dimensions. In PPIG 2002. (Found in http://ppig.org/workshops/14th-programme.html.) Exclusion reasons: Q1–2 [II.ajk]Based on the title alone, studies a specification, not programming, language.

1695. Anh Trinh (2011): X10 vs Java: Concurrency Constructs and Performance. . Master's thesis. http://scholarworks.sjsu.edu/etd_projects/203/ Exclusion reasons: Q5 [III.ajk]The arguably empirical part of this master's thesis focuses only on implementation efficiency.

1696. Matthew S. Tschantz & Michael D. Ernst (2005): Javari: adding reference immutability to Java. In Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. New York, NY, USA: ACM. OOPSLA '05. Pages 211-230. doi: 10.1145/1094811.1094828 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1697. M. Tukiainen (2001): Comparing two spreadsheet calculation paradigms: an empirical study with novice users. Interacting with Computers 13 (4). Pages 427-446. doi:10.1016/S0953-5438(00)00048-5 Exclusion reasons: Q1–2 [III.ajk]Spreadsheet programming in the sense of this article is excluded by our definition of a programming language.

1698. Franco Turini (1984): Magma2: a language oriented toward experiments in control. ACM Transactions on Programming Languages and Systems 6 (4). Pages 468-486. doi:10.1145/1780.1784 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1699. Lloyd D. Turnep, David Dahm, Warren Taylor & Richard E. Waychoff (1961): Letters to the editor: ALGOL 60 reply. Communications of the ACM 4 (9). Pages 365. doi:10.1145/366696.366700 Exclusion reasons: Q1–2 [III.ajk]This letter to the editor does not report a study.

1700. Aaron J. Turon & Claudio V. Russo (2011): Scalable join patterns. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 575-594. doi:10.1145/2048066.2048111 Exclusion reasons: Q1–2 [III.ajk]This article is about implementation, not language design.

1701. David Ungar & Sam S. Adams (2010): Harnessing emergence for manycore programming: early experience integrating ensembles, adverbs, and object-based inheritance. In SPLASH '10 Systems Programming Languages and Applications: Software for Humanity. Pages 19-26. doi:10.1145/1869542.1869546 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1702. Christopher Unkel & Monica S. Lam (2008): Automatic inference of stationary fields: a generalization of java's final fields. In Proc. 35th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 183-195. doi:10.1145/1328438.1328463 Exclusion reasons: Q1–2 [II.ajk]Implementation and program analysis techinique development.

1703. Asis Unyapoth & Peter Sewell (2001): Nomadic pict: correct communication infrastructure for mobile computation. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 116-127. doi:10.1145/360204.360214 Exclusion reasons: Q5 [II.ajk]Formal development,

1704. Kyle Usbeck & Jacob Beal (2011): An agent framework for agent societies. In Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE!'11, AOOPES'11, NEAT'11, \&\#38; VMIL'11. New York, NY, USA: ACM. Pages 201-212. doi:10.1145/2095050.2095082 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1705. R. E. Utman (1963): Suggestions on ALGOL 60 (ROME) issues. Communications of the ACM 6 (1). Pages 20-23. doi:10.1145/366193.366209 Exclusion reasons: Q5 [III.ajk]This language design memorandum does not aspire to empiricity.

1706. David Vadas & James R Curran (2005): Programming With Unrestricted Natural Language. In Proceedings of the Australasian Language Technology Workshop. Pages 191-199. Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1707. Tom Van Cutsem, Stijn Mostinckx, Wolfgang De Meuter, Jessie Dedecker & Theo D'Hondt (2004): On the Performance of SOAP in a Non-trivial Peer-to-Peer Experiment. Volume 3083.In Emmerich, Wolfgang and Wolf, Alexander (ed.) Component Deployment. Lecture Notes in Computer Science. Pages 205-218. doi:10.1007/978-3-540-24848-4_14 Exclusion reasons: Q5 [III.ajk]The evaluation reported in this article is clearly analytical, not empirical in nature.

1708. J. C. Van Vliet & H. M. Gladney (1985): An evaluation of tagging. Software: Practice and Experience 15 (9). Pages 823–837. doi:10.1002/spe.4380150902 Exclusion reasons: Q5 [III.ajk]This article is analytical, not empirical under our definition.

1709. Jean Vaucher, Guy Lapalme & Jacques Malenfant (1988): SCOOP Structured Concurrent Object Oriented Prolog. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 191-211. doi:10.1007/3-540-45910-3_12 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1710. Thomas P. Vayda (1995): Lessons from the battlefield. In OOPSLA '95: Proceedings of the tenth annual conference on Object-oriented programming systems, languages, and applications. Pages 439-452. doi:10.1145/217838.217881 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1711. Mandana Vaziri, Frank Tip & Julian Dolby (2006): Associating synchronization constraints with data in an object-oriented language. In Proc. 33nd ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 334-345. doi:10.1145/1111037.1111067 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1712. Mandana Vaziri, Frank Tip, Stephen Fink & Julian Dolby (2007): Declarative Object Identity Using Relation Types. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 54-78. doi:10.1007/978-3-540-73589-2_4 Exclusion reasons: Q1–2 [III.ajk]The included evaluation does not seem to focus on efficacy, more on feasibility.

1713. Mandana Vaziri, Frank Tip, Julian Dolby, Christian Hammer & Jan Vitek (2010): A Type System for Data-Centric Synchronization. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 304-328. doi:10.1007/978-3-642-14107-2_15 Exclusion reasons: Q5 [III.ajk]This article does not evaluate empirically its design decisions. The performance evaluations are empirical only so far as the implementations are concerned.

1714. Victor van der Veen, Nitish dutt-Sharma, Lorenzo Cavallaro & Herbert Bos (2012): Memory Errors: The Past, the Present, and the Future. Volume 7462.In Balzarotti, Davide and Stolfo, Salvatore and Cova, Marco (ed.) Research in Attacks, Intrusions, and Defenses.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 86-106. doi:10.1007/978-3-642-33338-5_5 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1715. Toon Verwaest, Camillo Bruni, Mircea Lungu & Oscar Nierstrasz (2011): Flexible object layouts: enabling lightweight language extensions by intercepting slot access. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 959-972. doi:10.1145/2048066.2048138 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1716. Iris Vessey & Ron Weber (1983): Some factors affecting program repair maintenance: an empirical study. Communications of the ACM 26 (2). Pages 128-134. doi:10.1145/358024.358057 Exclusion reasons: Q1–2 [II.ajk]No comparison, no PL design issue.

1717. Iris Vessey & Ron Weber (1986): Structured tools and conditional logic: an empirical investigation. Communications of the ACM 29 (1). Pages 48-57. doi:10.1145/5465.5470 Exclusion reasons: Q1–2 [II.ajk]No programming language relevance.

1718. Iris Vessey (1987): On matching programmers' chunks with program structures: An empirical investigation. International Journal of Man-Machine Studies 27 (1). Pages 65-89. doi:10.1016/S0020-7373(87)80044-5 Exclusion reasons: Q1–2 [II.ajk]No language design issues.

1719. Iris Vessey (1989): Toward a theory of computer program bugs: an empirical test. International Journal of Man-Machine Studies 30 (1). Pages 23-46. doi:10.1016/S0020-7373(89)80019-7 Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1720. Antonio Vetró, Federico Tomassetti, Marco Torchiano & Maurizio Morisio (2012): Language Interaction and Quality Issues: An Exploratory Study. In ESEM 2012 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement. doi:10.1145/2372251.2372309 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

1721. Santiago Vidal, Claudia Marcos, Alexandre Bergel & Gabriela Arévalo (2011): Memoization aspects: a case study. In Proceedings of the International Workshop on Smalltalk Technologies. New York, NY, USA: ACM. IWST '11. Pages 6:1-6:10. doi:10.1145/2166929.2166935 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1722. J.I. Villar, J. Juan, M.J. Bellido, J. Viejo, D. Guerrero & J. Decaluwe (2011): Python as a hardware description language: A case study. In Programmable Logic (SPL), 2011 VII Southern Conference on. Pages 117-122. doi:10.1109/SPL.2011.5782635 Exclusion reasons: Q1–2 [III.ajk]HDLs are not programming languages by our definition (they describe hardware, not computational processes that are parametrized over external input).

1723. Eelco Visser (2008): WebDSL: A Case Study in Domain-Specific Language Engineering. Volume 5235.In Lämmel, Ralf and Visser, Joost and Saraiva, João (ed.) Generative and Transformational Techniques in Software Engineering II.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 291-373. doi:10.1007/978-3-540-88643-3_7 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article does not aspire to empiricity.

1724. Jelena Vlasenko (2011): Exploring developer's tool path. In Proceedings of the ACM international conference companion on Object oriented programming systems languages and applications companion. New York, NY, USA: ACM. Pages 219-220. doi:10.1145/2048147.2048216 Exclusion reasons:

Q1–2 [II.ajk]Study of programmer practices, no language design issues.

1725. P. J. Voda (1982): Maple: a programming language and operating system. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 157-168. doi:10.1145/582153.582171 Exclusion reasons: Q1–2 [II.ajk]Language exposition

1726. Markus Voelter, Daniel Ratiu, Bernhard Schaetz & Bernd Kolb (2012): mbeddr: an extensible C-based programming language and IDE for embedded systems. In Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. New York, NY, USA: ACM. Pages 121-140. doi:10.1145/2384716.2384767 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity (experience reports are specifically excluded).

1727. Heiko Vogler (1991): Functional description of the contextual analysis in block-structured programming languages: a case study of tree transducers. Science of Computer Programming 16 (3). Pages 251-275. doi:10.1016/0167-6423(91)90009-M Exclusion reasons: Q1–2 [II.ajk]No PL design issue.

1728. Kris De Voider (1998): Aspect-Oriented Logic Meta Programming. In ECOOP'98 European Conference on Object-Oriented Programming Workshop Reader. Lecture Notes in Computer Science 1543. Pages 584-585. doi:10.1007/3-540-49255-0_123 Exclusion reasons: Q5 [III.ajk]Decision made based on http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.26.112, the PDF link in Science Direct being and having for a long time been broken (Springer has been notified many months ago). This article does not aspire to empiricity,

1729. Haris Volos, Andres Jaan Tack, Michael M. Swift & Shan Lu (2012): Applying transactional memory to concurrency bugs. In Proceedings of the seventeenth international conference on Architectural Support for Programming Languages and Operating Systems. New York, NY, USA: ACM. ASPLOS XVII. Pages 211-222. doi:10.1145/2150976.2150999 Exclusion reasons: Q1–2 [III.ajk]This article evaluates bug-fixing methodology that uses TM; however, there is no language design issue at play.

1730. DM Volpano & HE Dunsmore (1981): Problems with COBOL–Some Empirical Evidence. Computer Science Technical Reports 300 at Purdue University. http://docs.lib.purdue.edu/cstech/300/ Exclusion reasons: Q1–2 [III.ajk]This article reports an empirical study attempting to determine which COBOL constructs are problematic to programmers. It does not attempt to evaluate any specific design decision.

1731. Dennis M. Volpano & H. E. Dunsmore (1984): Empirical investigation of COBOL features. Information Processing & Management 20 (1-2). Pages 277-291. doi:10.1016/0306-4573(84)90060-8 Exclusion reasons: Q1–2 [II.ajk]Discovery of issues in Cobol; so far as this evaluates Cobol it does it without comparison.

1732. RA Volz, P Krishnan & R Theriault (1991): Distributed Ada: case study. Information and Software Technology 33 (4). Pages 292-300. doi:10.1016/0950-5849(91)90154-4 Exclusion reasons: Q1–2 [II.ajk]Describes an implementation; no direct PL design issue.

1733. Rob Von Behren, Jeremy Condit & Eric Brewer (2003): Why events are a bad idea (for high-concurrency servers). In Proc. HotOS IX: The 9th Workshop on Hot Topics in Operating Systems. Pages 19-24. http://static.usenix.org/events/hotos03/tech/vonbehren.html Exclusion reasons: Q5 [III.ajk]This article's only empirical aspect deals with implementation efficiency.

1734. Edward A. Voorhees (1958): Algebraic formulation of flow diagrams. Communications of the ACM 1 (6). Pages 4-8. doi:10.1145/368861.368869 Exclusion reasons: Q5 [III.ajk]This paper has no aspiration to empiricity.

1735. Jérôme Vouillon (2001): Combining subsumption and binary methods: an object calculus with views. In Proc. 28th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 290-303. doi:10.1145/360204.360233 Exclusion reasons: Q1–2 [II.ajk]Formal type-theoretical development.

1736. Oscar Waddell & R. Kent Dybvig (1999): Extending the scope of syntactic abstraction. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 203-215. doi:10.1145/292540.292559 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1737. P. Wadler (1987): Views: a way for pattern matching to cohabit with data abstraction. In Proc. 14th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 307-313. doi:10.1145/41625.41653 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1738. P. Wadler & S. Blott (1989): How to make ad-hoc polymorphism less ad hoc. In Proc. 16th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 60-76. doi:10.1145/75277.75283 Exclusion reasons: Q5 [II.ajk]Exposition and formal theoretical work.

1739. Philip Wadler (1992): The essence of functional programming. In Proc. 19th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 1-14. doi:10.1145/143165.143169 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1740. Robert A. Wagner (1970): Finiteness assumptions and intellectual isolation of computer scientists. Communications of the ACM 13 (12). Pages 759-760. (Included in cacm-3 (phase one) because a cursory read (because the title piqued interest; the paper was not suspected to be relevant) of the text makes it clear it is potentially relevant..) doi:10.1145/362814.362833 Exclusion reasons: Q5 [III.ajk]This is essentially a brief analytical comparison of Algol and Fortran.

1741. William M. Waite (1967): A language independent macro processor. Communications of the ACM 10 (7). Pages 433-440. doi:10.1145/363427.363458 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1742. Karen P. Walker & Stephen R. Schach (1996): Obstacles to Learning a Second Programming Language: An Empirical Study. Computer Science Education 7 (1). Pages 1-20. doi:10.1080/0899340960070101 Exclusion reasons: Q1–2 [II.ajk]Studies teaching tactics.

1743. Karen Pearce Walker (1998): Ada as a second programming language: an empirical study of assimilation of new language features. Nashville, TN, USA: Vanderbilt University. http://dl.acm.org/citation.cfm?id=926471 Exclusion reasons: Q1–2 [III.ajk]This pedagogical study takes the efficacy of language features as given and studies teaching them.

1744. Robert J. Walker, Shreya Rawal & Jonathan Sillito (2012): Do crosscutting concerns cause modularity problems?. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering. New York, NY, USA: ACM. FSE '12. Pages 49:1-49:11. doi:10.1145/2393596.2393654 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1745. David W. Wall (1982): Messages as active agents. In Proc. 9th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 34-39. doi:10.1145/582153.582157 Exclusion reasons: Q5 [III.ajk]This article introduces and analyzes a new manner of writing distributed programs. So far as there is a PL design issue, it is not empirically investigated.

1746. Peter L. Wallis (1980): External Representations of Objects of User-Defined Type. ACM Transactions on Programming Languages and Systems 2 (2). Pages 137-152. (There is a corrigendum in TOPLAS 3(1) p. 111 http://dx.doi.org/10.1145/357121.357130.) doi:10.1145/357094.357095 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1747. James F. Walsh (1992): Preliminary defect data from the iterative development of a large C++ program (experience report). In OOPSLA '92: conference proceedings on Object-oriented programming systems, languages, and applications. Pages 178-183. doi:10.1145/141936.141952 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1748. Yingxu Wang (2005): Psychological experiments on the cognitive complexities of fundamental control structures of software systems. In Cognitive Informatics, 2005. (ICCI 2005). Fourth IEEE Conference on. Pages 4-5. doi:10.1109/COGINF.2005.1532608 Exclusion reasons: Q1–2 [II.ajk]Metrics work; no evaluation of PL DDs.

1749. Cheng Wang & Daqing Hou (2008): An Empirical Study of Function Overloading in C++. In Source Code Analysis and Manipulation, 2008 Eighth IEEE International Working Conference on. Pages 47-56. doi:10.1109/SCAM.2008.25 Exclusion reasons: Q1–2 [II.ajk]Program corpus analysis.

1750. Keith Wansbrough & Simon Peyton Jones (1999): Once upon a polymorphic type. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 15-28. doi:10.1145/292540.292545 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1751. David H D Warren, Luis M. Pereira & Fernando Pereira (1977): Prolog - the language and its implementation compared with Lisp. In Proceedings of the 1977 symposium on Artificial intelligence and programming languages. New York, NY, USA: ACM. Pages 109-115. doi:10.1145/800228.806939 Exclusion reasons: Q5 [II.ajk]This article does not aspire to empiricity.

1752. Alessandro Warth, Milan Stanojević & Todd Millstein (2006): Statically scoped object adaptation with expanders. In OOPSLA '06: Proceedings of the 21st annual ACM SIGPLAN conference on Object-oriented programming systems, languages, and applications. Pages 37-56. doi:10.1145/1167473.1167477 Exclusion reasons: Q5 [III.ajk]This article includes an evaluation of the new construct by writing a program in both Java and the new language, and by using the new language to solve existing problems in Eclipse. These "case studies" have a markedly analytical flavour, as they try to show that the new construct is relevant, exploring the implications of the construct.

1753. Alessandro Warth, Yoshiki Ohshima, Ted Kaehler & Alan Kay (2011): Worlds: Controlling the Scope of Side Effects. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 179-203. doi:10.1007/978-3-642-22655-7_9 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity (the self-styled "case studies" are merely analytical examples).

1754. Tan Watanabe, Tsuneharu Ohsawa & Takaji Suzuki (1983): A simple database language for personal computers. Communications of the ACM 26 (9). Pages 646-653. doi:10.1145/358172.358181 Exclusion reasons: Q3 [III.ajk]This article describes, among other things, a simple empirical comparison of Micro-MUMPS to COBOL and assembly. However, this comparison is discussed summarily and its validity cannot be assessed from the discussion.

1755. Richard C. Waters (1983): User Format Control in a LISP Prettyprinter. ACM Transactions on Programming Languages and Systems 5 (4). Pages 513-531. doi:10.1145/69575.357225 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1756. Richard C. Waters (1984): Expressional loops. In Proc. 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL).

Pages 1-10. doi:10.1145/800017.800511 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1757. Philippe Weber & Daniel Taupin (1995): EXPHER (EXperimental PHysics ERror analysis): a Declaration Language and a Program Generator for the Treatment of Experimental Data. J. Phys. III France 5 (5). Pages 605-622. doi:10.1051/jp3:1995149 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1758. Stephen Weeks & Matthias Felleisen (1993): On the orthogonality of assignments and procedures in Algol. In Proc. 20th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 57-70. doi:10.1145/158511.158523 Exclusion reasons: Q1–2 [II.ajk]Formal theoretical work; no PL design issue.
1759. Dasarath Weeratunge, Xiangyu Zhang & Suresh Jaganathan (2011): Accentuating the positive: atomicity inference and enforcement using correct executions. In Proceedings of the 2011 ACM international conference on Object oriented programming systems languages and applications. New York, NY, USA: ACM. Pages 19-34. doi:10.1145/2048066.2048071 Exclusion reasons: Q1–2 [III.ajk]There does not seem to be a language design issue here.
1760. Ben Wegbreit (1974): The treatment of data types in EL1. Communications of the ACM 17 (5). Pages 251-264. doi:10.1145/360980.360992 Exclusion reasons: Q1–2 [II.ajk]Single language exposition
1761. Peter Wegner (1962): Communications between independently translated blocks. Communications of the ACM 5 (7). Pages 376-381. doi:10.1145/368273.368279 Exclusion reasons: Q5 [III.ajk]This brief analytical paper has no aspiration to empiricity.
1762. Eberhard Wegner (1973): Tree-structured programs. Communications of the ACM 16 (11). Pages 704-705. doi:10.1145/355611.362547 Exclusion reasons: Q1–2 [II.ajk]Pogramming style discussion, no PL design issue.
1763. Peter Wegner (1983): On the unification of data and program abstraction in Ada. In Proc. 10th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 256-264. doi:10.1145/567067.567091 Exclusion reasons: Q5 [III.ajk]This analytical paper does not aspire to empiricity.
1764. Peter Wegner & Stanley B. Zdonik (1988): Inheritance as an Incremental Modification Mechanism or What Like Is and Isn't Like. In Proc. ECOOP'88 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 322. Pages 55-77. doi:10.1007/3-540-45910-3_4 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1765. J. H. Wegstein (1959): From formulas to computer oriented language. Communications of the ACM 2 (3). Pages 6-8. doi:10.1145/368300.368318 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.
1766. J. H. Wegstein & W. W. Youden (1962): A string language for symbol manipulation based on ALGOL 60. Communications of the ACM 5 (1). Pages 54-61. doi:10.1145/366243.366745 Exclusion reasons: Q5 [III.ajk]This language exposition does not report an empirical study.
1767. Stefan Wehr, Ralf Lämmel & Peter Thiemann (2007): JavaGI: Generalized Interfaces for Java. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609 . Pages 347-372. doi:10.1007/978-3-540-73589-2_17 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1768. Stefan Wehr & Peter Thiemann (2011): JavaGI: The Interaction of Type Classes with Interfaces and Inheritance. ACM Transactions on Programming Languages and Systems 33 (4). Pages 12:1–12:83. doi:10.1145/1985342.1985343 Exclusion reasons: Q1–2 [III.ajk]The arguably empirical evaluation in this paper focuses on implementation non-efficiency.
1769. Reinhold P. Weicker (1984): Dhrystone: a synthetic systems programming benchmark . Communications of the ACM 27 (10). Pages 1013-1030. doi:10.1145/358274.358283 Exclusion reasons: Q1–2 [III.ajk]This article presents a benchmark program for evaluation of computers. It has no bearing on language design issues.
1770. T. G. Weidner (1979): CHAMIL A Case Study in Microprogramming Language Design. In Compcon Fall 79. Proceedings. Pages 79-83. doi:10.1109/CMPCON.1979.729087 Exclusion reasons: Q1–2 [II.ajk]Discusses a particular language design process generally, based on the abstract.
1771. Thomas G. Weidner (1980): CHAMIL: a case study in microprogramming language design. SIGPLAN Notices 15 (1). Pages 156–166. doi:10.1145/954127.954145 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1772. William Weihl & Barbara Liskov (1985): Implementation of resilient, atomic data types. ACM Transactions on Programming Languages and Systems 7 (2). Pages 244-269. doi:10.1145/3318.3319 Exclusion reasons: Q5 [III.ajk]This article has no aspiration for empiricity.
1773. W. E. Weihl (1989): Local atomicity properties: modular concurrency control for abstract data types. ACM Transactions on Programming Languages and Systems 11 (2). Pages 249-282. doi:10.1145/63264.63518 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1774. William E. Weihl (1990): Linguistic support for atomic data types. ACM Transactions on Programming Languages and Systems 12 (2). Pages 178-202. doi:10.1145/78942.78944 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
1775. Gerald M Weinberg (1971): The psychology of computer programming. New York: Van Nostrand Reinhold. Exclusion reasons: Q5 [III.ajk]This book does not summarise or consolidate empirical research on the efficacy of PL design decisions. It does offer expert advice, however.
1776. Gerald M. Weinberg, Dennis P. Geller & Thomas W. S. Plum (1975): IF-THEN-ELSE considered harmful. SIGPLAN Notices 10 (8). Pages 34-44. doi:10.1145/956028.956032 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1777. Arla E. Weinert (1967): A SIMSCRIPT-FORTRAN case study. Communications of the ACM 10 (12). Pages 784-792. doi:10.1145/363848.363862 Exclusion reasons: Q3 [III.ajk]This article reports a study in which two programs were (presumably) written in SIMSCRIPT and FORTRAN, respectively, implementing the same specification, and those programs are then analyzed in various ways. While there may be some empiricity in the production of the programs, the process is inadequately described for any assesessment of methodological quality to be possible.
1778. Adam Welc, Suresh Jagannathan & Antony Hosking (2005): Safe futures for Java. In OOPSLA '05: Proceedings of the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 439-453. doi:10.1145/1094811.1094845 Exclusion reasons: Q1–2 [III.ajk]This article includes a section reporting "experiments" about the new construction. These experiments are straightforward performance bechmarks using (apparently) artificial payload programs. It is hard to see how this evaluates the design decisions made.
1779. PH Welch (1998): Java threads in light of Occam/CSP. In Proc WoTUG 21. Exclusion reasons: Q3 [II.ajk]This is a mere abstract of a tutorial, not a full published study report.
1780. Peter H. Welch & Jan B. Pedersen (2010): Santa Claus: Formal analysis of a process-oriented solution. ACM Transactions on Programming Languages and Systems 32 (4). doi:10.1145/1734206.1734211 Exclusion reasons: Q1–2 [II.ajk]Program design, not language design, issue.
1781. A. J. Wellings, B. Johnson, B. Sanden, J. Kienzle, T. Wolf & S. Michell (2000): Integrating object-oriented programming and protected objects in Ada 95. ACM Transactions on Programming Languages and Systems 22 (3). Pages 506-539. doi:10.1145/353926.353938 Exclusion reasons: Q5 [III.ajk]This article has no aspiration to empiricity.
1782. Mark B. Wells (1963): Recent improvements in MADCAP. Communications of the ACM 6 (11). Pages 674-678. doi:10.1145/368310.368389 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1783. J. Welsh, W. J. Sneeringer & C. A. R. Hoare (1977): Ambiguities and insecurities in pascal. Software: Practice and Experience 7 (6). Pages 685-696. doi:10.1002/spe.4380070604 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1784. C. S. Wetherell (1982): Error Data Values in the Data-Flow Language VAL. ACM Transactions on Programming Languages and Systems 4 (2). Pages 226-238. doi:10.1145/357162.357167 Exclusion reasons: Q5 [III.ajk]This paper does not aspire to empiricality.
1785. M. W. Whitelaw (1985): Some ramifications of the EXIT statement in loop control. SIGPLAN Notices 20 (8). Pages 99-106. doi:10.1145/988346.988361 Exclusion reasons: Q1–2 [III.ajk]This article does not aspire to empiricity.
1786. K. N. Whitley (1997): Visual Programming Languages and the Empirical Evidence For and Against. Journal of Visual Languages & Computing 8 (1). Pages 109-142. doi:10.1006/jvlc.1996.0030 Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded as per our PL definition.
1787. Kirsten N. Whitley, Laura R. Novick & Doug Fisher (2006): Evidence in favor of visual representation for the dataflow paradigm: An experiment testing LabVIEW's comprehensibility. International Journal of Human-Computer Studies 64 (4). Pages 281-303. doi:10.1016/j.ijhcs.2005.06.005 http://www.sciencedirect.com/science/article/pii/S1071581905001163 Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded.
1788. Kirsten N. Whitley, Laura R. Novick & Doug Fisher (2006): Evidence in favor of visual representation for the dataflow paradigm: An experiment testing LabVIEW's comprehensibility. International Journal of Human-Computer Studies 64 (4). Pages 281-303. doi:10.1016/j.ijhcs.2005.06.005 Exclusion reasons: Q1–2 [II.ajk]Comparing textual to graphical language; since a graphical language is excluded per our definition of a programming language, there is no (relevant) comparison.
1789. JON WHITTLE & ANDREW CUMMING (2000): Evaluating environments for functional programming. International Journal of Human-Computer Studies 52 (5). Pages 847-878. doi:10.1006/ijhc.1999.0356 Exclusion reasons: Q1–2 [II.ajk]Deals with programming environments, not languages.
1790. Brian A. Wichmann (1984): Is Ada too big? A designer answers the critics. Communications of the ACM 27 (2). Pages 98-103. doi:10.1145/69610.69613 Exclusion reasons: Q1–2 [II.ajk]No comparison.
1791. Susan Wiedenbeck (1986): Beacons in computer program comprehension. International Journal of Man-Machine Studies 25 (6). Pages 697-709. doi:10.1016/S0020-7373(86)80083-9 Exclusion reasons: Q1–2 [II.ajk]Studies a programming comprehension theory, no language design issue.
1792. Susan Wiedenbeck (1989): Learning iteration and recursion from examples. International Journal of Man-Machine Studies 30 (1). Pages 1-22. doi:10.1016/S0020-7373(89)80018-5 Exclusion reasons: Q1–2 [III.ajk]This article investigates teaching/learning approaches and does not evaluate any

language design decisions.
1793. Susan Wiedenbeck (1991): The initial stage of program comprehension. International Journal of Man-Machine Studies 35 (4). Pages 517-540. doi: 10.1016/S0020-7373(05)80090-2 Exclusion reasons: Q1–2 [II.ajk]No language design issues.
1794. R S Wiener (1987): Object-oriented programming in C++\—a case study. SIGPLAN Notices 22 (6). Pages 59-68. doi:10.1145/24900.24906 Exclusion reasons: Q5 [III.ajk]This article consists almost exclusively of a program listing, along with brief comments claiming that it demonstrates C++'s efficacy. However, at best this is an analytical study and not empirical.
1795. David S. Wile (1983): Program developments: formal explanations of implementations. Communications of the ACM 26 (11). Pages 902-911. doi: 10.1145/182.358443 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1796. Jack C. Wileden, Alexander L. Wolf, Charles D. Fisher & Peri L. Tarr (1988): Pgraphite: an experiment in persistent typed object management. In Proceedings of the third ACM SIGSOFT/SIGPLAN software engineering symposium on Practical software development environments. Pages 130-142. doi:10.1145/64135.65016 Exclusion reasons: Q3 [III.ajk]This article does not, in general, aspire to empiricity; there is one section that approaches empirical work but it's too vaguely described to qualify.
1797. M. V. Wilkes (1964): Constraint-type statements in programming languages. Communications of the ACM 7 (10). Pages 587-588. doi:10.1145/364888.364967 Exclusion reasons: Q5 [III.ajk]This brief article does not aspire to empiricity.
1798. Hernán Wilkinson, Máximo Prieto & Luciano Romeo (2005): Arithmetic with measurements on dynamically-typed object-oriented languages. In OOPSLA '05: Companion to the 20th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications. Pages 292-300. doi:10.1145/1094855.1094964 Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.
1799. R Wilkinson, B Hegner & C D Jones (2010): Usage of the Python programming language in the CMS experiment. Journal of Physics: Conference Series 219 (4). Pages 042026. doi:10.1088/1742-6596/219/4/042026 Exclusion reasons: Q1–2 [II.ajk]No PL design issue; in any event, no comparison language.
1800. H. J. Will (1983): ACL: a language specific for auditors. Communications of the ACM 26 (5). Pages 128-134. doi:10.1145/69586.358138 Exclusion reasons: Q1–2 [II.ajk]Language exposition.
1801. JH Williams (1977): An Evaluation of Process and Experiment Automation Realtime Language (PEARL). at DTIC Document. http://oai.dtic.mil/oai/oai?verb=getRecord&metadataPrefix=html&identifier=ADA037641 Exclusion reasons: Q1–2 [II.ajk]No comparison.
1802. J. H. Williams & E. L. Wimmers (1988): Sacrificing simplicity for convenience: Where do you draw the line?. In Proc. 15th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 169-179. doi:10.1145/73560.73575 Exclusion reasons: Q5 [II.ajk]Theoretical investigation.
1803. Fleur L. Williams & Gordon B. Steven (1990): How useful are complex instructions? A case study using the M68000. Microprocessing and Microprogramming 29 (4). Pages 247-259. doi:10.1016/0165-6074(90)90343-8 Exclusion reasons: Q1–2 [II.ajk]Studies a machine language as a compiler target, not as a programming language by our definition.
1804. MARIAN G. WILLIAMS & J.NICHOLAS BUEHLER (1999): Comparison of visual and textual languages via task modeling. International Journal of Human-Computer Studies 51 (1). Pages 89-115. doi:10.1006/ijhc.1999.0270 Exclusion reasons: Q1–2 [II.ajk]Visual languages are excluded.
1805. Darren Willis, David J. Pearce & James Noble (2006): Efficient Object Querying for Java. In Proc. ECOOP 2006 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4067. Pages 28-49. doi:10.1007/11785477_3 Exclusion reasons: Q5 [III.ajk]This analytical-constructive article includes an evaluation section in which specifically chosen simple queries are tested, pitting against each other the constructed automatic translator and two hand-crafter translations, for each query, measuring performance. The only contingency in this setup is the translator and the translations. As far as evaluating the language design decision goes, this "experiment" answers a "can-our-construct-do-it" (as opposed to "how-does-it-perform-in-the-real-world") question and is thus fundamentally analytic, not empirical.
1806. Gregory V. Wilson, Jonathan Schaeffer & Duane Szafron (1993): Enterprise in context: assessing the usability of parallel programming environments. In Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: distributed computing - Volume 2.IBM Press. CASCON '93. Pages 999-1010. http://dl.acm.org/citation.cfm?id=962367.962403 Exclusion reasons: Q5 [III.ajk]This article does not report an empirical study.
1807. Amanda Wilson & David C. Moffat (2010): Evaluating Scratch to introduce younger schoolchildren to programming. In PPIG 2010. (Found in http://ppig2010.org/index.php?title=Program.) Exclusion reasons: Q1–2 [III.ajk]Full text found at http://www.ppig.org/papers/22nd-Teach-6.pdf (the links in http://www.ppig.org/workshops/22nd-programme.html are scrambled). This article discusses a visual language, which is excluded by our definition of PLs.
1808. J. A. Winkler, III & Edwin Towster (1977): A human factors study of structured programming techniques. In Proceedings of the 15th annual Southeast regional conference. New York, NY, USA: ACM. ACM-SE 15. Pages 402-408. doi:10.1145/1795396.1795452 Exclusion reasons: Q1–2 [III.ajk]This article presents a plan for an ongoing study and does not report results.
1809. J. F. H. Winkler (1984): Some improvements of ISO-Pascal. SIGPLAN Notices 19 (9). Pages 49-62. doi:10.1145/948596.948604 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1810. J. F. H. Winkler (1984): Some improvements of ISO-Pascal. SIGPLAN Notices 19 (7). doi:10.1145/988574.988582 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity
1811. Robert I. Winner (1984): Unassigned objects. ACM Transactions on Programming Languages and Systems 6 (4). Pages 449-467. doi:10.1145/1780.1785 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1812. Terry Winograd (1979): Beyond programming languages. Communications of the ACM 22 (7). Pages 391-401. doi:10.1145/359131.359133 Exclusion reasons: Q5 [III.ajk]This discussion paper does no empirical evaluation.
1813. Allen Wirfs-Brock & Brian Wilkerson (1988): A overview of modular smalltalk. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA'88). Pages 123-134. doi:10.1145/62083.62095 Exclusion reasons: Q1–2 [II.ajk]Language exposition
1814. Niklaus Wirth (1963): A generalization of ALGOL. Communications of the ACM 6 (9). Pages 547-554. doi:10.1145/367593.367619 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.
1815. Niklaus Wirth & C. A. R. Hoare (1966): A contribution to the development of ALGOL. Communications of the ACM 9 (6). Pages 413-432. doi: 10.1145/365696.365702 Exclusion reasons: Q1–2 [III.ajk]This article is a language exposition.
1816. Niklaus Wirth & Helmut Weber (1966): EULER: a generalization of ALGOL, and its formal definition: Part II. Communications of the ACM 9 (2). Pages 89-99. doi:10.1145/365170.365202 Exclusion reasons: Q1–2 [II.ajk]Language exposition, no comparison.
1817. Niklaus Wirth & Helmut Weber (1966): EULER: a generalization of ALGOL and it formal definition: Part 1. Communications of the ACM 9 (1). Pages 13-25. doi:10.1145/365153.365162 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1818. N. Wirth (1975): An assessment of the programming language pascal. Software Engineering, IEEE Transactions on SE-1 (2). Pages 192-198. doi: 10.1109/TSE.1975.6312839 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1819. Niklaus Wirth (1985): From programming language design to computer construction. Communications of the ACM 28 (2). Pages 160-164. doi: 10.1145/2786.2789 Exclusion reasons: Q1–2 [III.ajk]This Turing award lecture does not report a study.
1820. N. Wirth (1988): Type extensions. ACM Transactions on Programming Languages and Systems 10 (2). Pages 204-214. doi:10.1145/42190.46167 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1821. Robert Wise (1988): Experimental evaluation of a parallel programming language. . MSc at University of Virginia. http://en.scientificcommons.org/3996377 Exclusion reasons: Q5 [III.ajk]This master's thesis compares two languages by having the thesis author reimplement certain challenge programs in one of the languages, the original implementations being in the other language, and then comparing the languages analytically and in performance. Since the only clearly empirical comparison is of performance, it's not efficacy relevant.
1822. C. Wohlin (2010): Is prior knowledge of a programming language important for software quality?. In ISESE 2002 Proceedings: 2002 International Symposium on Empirical Software Engineering. Pages 27-34. doi:10.1109/ISESE.2002.1166922 Exclusion reasons: Q1–2 [II.ajk]No relevance to programming language design.
1823. Wayne Wolf (1989): A practical comparison of two object-oriented languages. Software, IEEE 6 (5). Pages 61-68. doi:10.1109/52.35590 Exclusion reasons: Q1–2 [III.ajk]This analytical article does not aspire to empiricity.
1824. Stephen Wolfram (1985): Symbolic mathematical computation. Communications of the ACM 28 (4). Pages 390-394. doi:10.1145/3341.3347 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.
1825. Derick Wood (1969): A few more trouble spots in ALGOL 60. Communications of the ACM 12 (5). Pages 247-248. doi:10.1145/362946.362957 Exclusion reasons: Q5 [III.ajk]This letter to the editor does not report an empirical study.
1826. Murray Wood, John Daly, James Miller & Marc Roper (1999): Multi-method research: An empirical investigation of object-oriented technology. Journal of Systems and Software 48 (1). Pages 13-26. doi:10.1016/S0164-1212(99)00042-4 Exclusion reasons: Q1–2 [III.ajk]This article reports on a series of

empirical studies and evaluates their methodological approach. However, there was no PL design issue present.

1827. D. C. Wood (2000): An Experiment with Recursion in occam. In Communicating Process Architectures 2000: WoTUG-23: Proceedings of the 23rd World Occam and Transputer User Group Technical Meeting: 10-13 September 2000, Canterbury, United Kingdom.Ios. Pages 193-204. http://wotug.kent.ac.uk/paperdb/show_proc.php?f=4&num=18 Exclusion reasons: Q1–2 [II.ajk]This article demonstrates that something is possible in occam; it does not evaluate any design decisions.

1828. S. N. Woodfield, H. E. Dunsmore & V. Y. Shen (1981): The effect of modularization and comments on program comprehension. In Proceedings of the 5th international conference on Software engineering. Piscataway, NJ, USA: IEEE Press. ICSE '81. Pages 215-223. http://dl.acm.org/citation.cfm?id=800078.802534 Exclusion reasons: Q1–2 [III.ajk]This article studies programming style with no clear language design issue at play, either literally or through an analogue.

1829. John D. Woolley, Leland R. Miller & Charles M. Bernstein (1976): LINUS: an experiment in language preprocessing. SIGPLAN Notices 11 (9). Pages 38-48. doi:10.1145/987500.987506 Exclusion reasons: Q5 [III.ajk]This article does not report empirical evidence.

1830. Tobias Wrigstad, Filip Pizlo, Fadi Meawad, Lei Zhao & Jan Vitek (2009): Loci: Simple Thread-Locality for Java. In Proc. ECOOP 2009 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5653. Pages 445-469. doi:10.1007/978-3-642-03013-0_21 Exclusion reasons: Q5 [III.ajk]This article presents and analyzes an statically verifiable annotation system for thead-locality in Java, with no aspiration to empiricity.

1831. Tobias Wrigstad, Francesco Zappa Nardelli, Sylvain Lebresne, Johan Östlund & Jan Vitek (2010): Integrating typed and untyped code in a scripting language. In Proc. 37th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 377-388. doi:10.1145/1706299.1706343 Exclusion reasons: Q1–2 [III.ajk]This article introduces and analyses a new language. It also includes a section discussing the authors' experience with the new language, but there is no comparison to another language of any note. Thus, there is no efficacy issue here.

1832. Quanfeng Wu & John R. Anderson (1993): Strategy choice and change in programming. International Journal of Man-Machine Studies 39 (4). Pages 579-598. doi:10.1006/imms.1993.1074 Exclusion reasons: Q1–2 [III.ajk]This article studies programming strategies; it does not evaluate language design decisions.

1833. Q. Wu & J. R. Anderson (1993): Knowledge Transfer among Programming Languages . In PPIG 1993. (Found in http://ppig.org/workshops/5th-programme.html.) Exclusion reasons: Q1–2 [III.ajk]This article reports a study on transfer from language to another. It has very little PL design relevance.

1834. W. A. Wulf, D. B. Russell & A. N. Habermann (1971): BLISS: a language for systems programming. Communications of the ACM 14 (12). Pages 780-790. doi:10.1145/362919.362936 Exclusion reasons: Q1–2 [II.ajk]Single language exposition

1835. W. Wulf & Mary Shaw (1973): Global variable considered harmful. SIGPLAN Notices 8 (2). Pages 28-34. doi:10.1145/953353.953355 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1836. Eric Van Wyk, Lijesh Krishnan, Derek Bodin & August Schwerdfeger (2007): Attribute Grammar-Based Language Extensions for Java. In Proc. ECOOP 2007 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 4609. Pages 575-599. doi:10.1007/978-3-540-73589-2_27 Exclusion reasons: Q1–2 [II.ajk]Discusses a language implementation tool.

1837. Eric Van Wyk & Eric Johnson (2007): Composable Language Extensions for Computational Geometry: A Case Study. In Hawaii International Conference on System Sciences. Los Alamitos, CA, USA: IEEE Computer Society. Pages 258c. doi:10.1109/HICSS.2007.139 Exclusion reasons: Q5 [III.ajk]Despite its title, this article does not aspire to empiricity.

1838. Hongwei Xi & Frank Pfenning (1999): Dependent types in practical programming. In Proc. 26th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 214-227. doi:10.1145/292540.292560 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1839. Hongwei Xi, Chiyan Chen & Gang Chen (2003): Guarded recursive datatype constructors. In Proc. 30th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 224-235. doi:10.1145/604131.604150 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1840. Guoqing Xu, Dacong Yan & Atanas Rountev (2012): Static Detection of Loop-Invariant Data Structures. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 738-763. doi:10.1007/978-3-642-31057-7_32 Exclusion reasons: Q1–2 [II.ajk]Implementation technique.

1841. Hong Yul Yang, E. Tempero & H. Melton (2008): An Empirical Study into Use of Dependency Injection in Java. In Software Engineering, 2008. ASWEC 2008. 19th Australian Conference on. Pages 239-247. doi:10.1109/ASWEC.2008.4483212 Exclusion reasons: Q1–2 [II.ajk]This article does not evaluate any language design decisions.

1842. Jean Yang, Kuat Yessenov & Armando Solar-Lezama (2012): A language for automatically enforcing privacy policies. In Proceedings of the 39th annual ACM SIGPLAN-SIGACT symposium on Principles of programming languages.ACM. POPL '12. Pages 85-96. doi:10.1145/2103656.2103669 Exclusion reasons: Q5 [III.ajk]Evaluation is analytical.

1843. Lynn D. Yarbrough (1962): Input data organization in FORTRAN. Communications of the ACM 5 (10). Pages 508-509. doi:10.1145/368959.368976 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1844. Wei Jen Yeh & Michal Young (1994): Re-designing tasking structures of Ada programs for analysis: A case study. Software Testing, Verification and Reliability 4 (4). Pages 223-253. doi:10.1002/stvr.4370040404 Exclusion reasons: Q1–2 [II.ajk]Program design study; no PL design issue.

1845. Daniel M. Yellin & Robert E. Strom (1991): INC: a language for incremental computations. ACM Transactions on Programming Languages and Systems 13 (2). Pages 211-236. doi:10.1145/103135.103137 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1846. Shaula Yemini & Daniel M. Berry (1985): A modular verifiable exception handling mechanism. ACM Transactions on Programming Languages and Systems 7 (2). Pages 214-243. doi:10.1145/3318.3320 Exclusion reasons: Q1–2 [II.ajk]Language exposition, formal development.

1847. M.Y.-M. Yen & R.W. Scamell (1993): A human factors experimental comparison of SQL and QBE. Software Engineering, IEEE Transactions on 19 (4). Pages 390-409. doi:10.1109/32.223806 Exclusion reasons: Q1–2 [II.ajk]Studies query languages, not PLs.

1848. Yasuhiko Yokote & Mario Tokoro (1987): Experience and evolution of concurrent Smalltalk. In Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications (OOPSLA87). Pages 406-415. doi:10.1145/38765.38844 Exclusion reasons: Q1–2 [III.ajk]This article does not empirically evaluate any language design decisions.

1849. SeungWook Yoo, Kyoung-A Kim, Yong Kim, YongChul Yeum, Susumu Kanemune & WonGyu Lee (2006): Empirical Study of Educational Programming Language for K12: Between Dolittle and Visual Basic. International Journal of Computer Science and Network Security 6 (6). Pages 118-123. http://paper.ijcsns.org/07_book/html/200606/200606020.html Exclusion reasons: Q1–2 [III.ajk]This article presents an empirical study in which two programming languages were used to teach high-school students programming, and their measured learning outcomes were compared. Thus, it evaluated the languages as teaching vehicles, not as programmer tools, and thus the study is not relevant to efficacy as we use the term here.

1850. Richard M. Yoo, Yang Ni, Adam Welc, Bratin Saha, Ali-Reza Adl-Tabatabai & Hsien-Hsin S. Lee (2008): Kicking the tires of software transactional memory: why the going gets tough. In Proceedings of the twentieth annual symposium on Parallelism in algorithms and architectures. New York, NY, USA: ACM. SPAA '08. Pages 265–274. doi:10.1145/1378533.1378582 Exclusion reasons: Q1–2 [III.ajk]This article only evaluates implementation efficiency.

1851. Matsuki Yoshino (1986): APL as a prototyping language: case study of a compiler development project. In Proceedings of the international conference on APL. New York, NY, USA: ACM. Pages 235-242. doi:10.1145/22415.22042 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

1852. Jia-Huai You & P. A. Subrahmanyam (1986): Equational logic programming: an extension to equational programming. In Proc. 13th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages (POPL). Pages 209-218. doi:10.1145/512644.512663 Exclusion reasons: Q1–2 [II.ajk]Language exposition, theoretical study.

1853. S. J. Young & C. Proctor (1986): UFL: An Experimental Frame Language Based on Abstract Data Types. The Computer Journal 29 (4). Pages 340-347. doi:10.1093/comjnl/29.4.340 Exclusion reasons: Q1–2 [II.ajk]Language exposition.

1854. Edward A. Youngs (1974): Human Errors in Programming. International Journal of Man-Machine Studies 6 (3). Pages 361-376. doi:10.1016/S0020-7373(74)80027-1 Exclusion reasons: Q1–2 [III.ajk]This article does not in any nontrivial sense evaluate any language design decisions for efficacy.

1855. Tristyanti Yusnitasari, Naeli Umniati & Deni Deni (2009): ANALYZING PROGRAMMING LANGUAGE GENTEE WITH C LANGUAGE (Case Study APPLICATION PROGRAM N FAKTORIAL). Informatics and Computing 14 (1). http://ejournal.gunadarma.ac.id/index.php/infokom/article/view/73 Exclusion reasons: Q3 [III.ajk]PDF not available; journal asked for assistance on 2012-12-12 with no change.

1856. Shin'ichi Yuta, Shooji Suzuki & Shigeki Iida (1993): Implementation of a small size experimental self-contained autonomous robot — sensors, vehicle control, and description of sensor based behavior. Volume 190.In Chatila, Raja and Hirzinger, Gerd (ed.) Experimental Robotics II.Springer Berlin / Heidelberg. Lecture Notes in Control and Information Sciences. Pages 344-358. doi:10.1007/BFb0036150 Exclusion reasons: Q1–2 [II.ajk]No relevance to programming language design.

1857. Razieh Nokhbeh Zaeem & Sarfraz Khurshid (2010): Contract-Based Data Structure Repair Using Alloy. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6183. Pages 577-598. doi:10.1007/978-3-642-14107-2_27 Exclusion reasons: Q1–2 [III.ajk]This article presents and evaluates a fault recovery system. There is no PL design issue involved.

1858. Bradley T. Vander Zanden, Richard Halterman, Brad A. Myers, Rich McDaniel, Rob Miller, Pedro Szekely, Dario A. Giuse & David Kosbie (2001): Lessons learned about one-way, dataflow constraints in the Garnet and Amulet graphical toolkits. ACM Transactions on Programming Languages and Systems 23 (6). Pages 776-796. doi:10.1145/506315.506318 Exclusion reasons: Q1–2 [II.ajk]Implementation issues

1859. Dmitrijs Zaparanuks & Matthias Hauswirth (2011): The Beauty and the Beast: Separating Design from Algorithm. In Proc. ECOOP 2010 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 6813. Pages 27-51. doi:10.1007/978-3-642-22655-7_3 Exclusion reasons: Q1–2 [II.ajk]Presents and evaluates a program analysis method.

1860. W. A. Zaremba (1965): On ALGOL I/O conventions. Communications of the ACM 8 (3). Pages 167-169. doi:10.1145/363791.363807 Exclusion reasons: Q5 [III.ajk]This short article does not aspire to empiricity.

1861. Pamela Zave (1984): The operational versus the conventional approach to software development. Communications of the ACM 27 (2). Pages 104-118. doi:10.1145/69610.357982 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate any language design decisions.

1862. Marvin V. Zelkowitz (1977): Effects of structured programming on PL/I programmers. Software: Practice and Experience 7 (6). Pages 793-795. doi: 10.1002/spe.4380070613 Exclusion reasons: Q1–2 [II.ajk]This article studies actual programming practice in a single language; there is no language design issue at play.

1863. Marvin V. Zelkowitz (1980): A case study in rapid prototyping. Software: Practice and Experience 10 (12). Pages 1037-1042. doi:10.1002/spe.4380101209 Exclusion reasons: Q1–2 [III.ajk]This article reports a single case of a program design process in which an interpreter for a language was first written in SNOBOL and then translated manually to Pascal for efficiency. It may say something about the usefulness of that process, but it certainly does not evaluate any language design decisions.

1864. H. Zemanek (1966): Semiotics and programming languages. Communications of the ACM 9 (3). Pages 139-143. doi:10.1145/365230.365249 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate language design decisions.

1865. Andreas Zendler (2001): A Preliminary Software Engineering Theory as Investigated by Published Experiments. Empirical Software Engineering 6 (2). Pages 161-180. doi:10.1023/A:1011489321999 Exclusion reasons: Q1–2 [III.ajk]This article does not evaluate (nor summarise or consolidate research evaluating) language design decisions.

1866. Matthias Zenger (2002): Type-Safe Prototype-Based Component Evolution. In Proc. ECOOP 2002 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 2374. Pages 470-497. doi:10.1007/3-540-47993-7_20 Exclusion reasons: Q1–2 [II.ajk]Formal discussion.

1867. Chao Zhang, Chenning Xie, Zhiwei Xiao & Haibo Chen (2011): Evaluating the Performance and Scalability of MapReduce Applications on X10. Volume 6965.In Temam, Olivier and Yew, Pen-Chung and Zang, Binyu (ed.) Advanced Parallel Processing Technologies.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 46-57. doi:10.1007/978-3-642-24151-2_4 Exclusion reasons: Q1–2 [III.ajk]"Our result showed that X10 is a powerful programming language to run MapReduce-style applications on clusters." (p.57). The quote shows that the study is focused on an nonempirical question (can X10 do it) rather than an empirical one (how well does X10 do it). The article contains performance measurements that do not make an efficacy issue.

1868. Lukasz Ziarek, Adam Welc, Ali-Reza Adl-Tabatabai, Vijay Menon, Tatiana Shpeisman & Suresh Jagannathan (2008): A Uniform Transactional Execution Environment for Java. In Proc. ECOOP 2008 European Conference on Object-Oriented Programming. Lecture Notes in Computer Science 5142. Pages 129-154. doi:10.1007/978-3-540-70592-5_7 Exclusion reasons: Q1–2 [II.ajk]No language design issue.

1869. Yoav Zibin, Alex Potanin, Paley Li, Mahmood Ali & Michael D. Ernst (2010): Ownership and immutability in generic Java. In OOPSLA '10: Proceedings of the ACM international conference on Object oriented programming systems languages and applications. Pages 598-617. doi:10.1145/1869459.1869509 Exclusion reasons: Q5 [III.ajk]This article does not, despite the "case study" included, aspire to empiricity.

1870. Yoav Zibin, David Cunningham, Igor Peshansky & Vijay Saraswat (2012): Object Initialization in X10. Volume 7313.In Noble, James (ed.) ECOOP 2012 – Object-Oriented Programming.Springer Berlin / Heidelberg. Lecture Notes in Computer Science. Pages 207-231. doi:10.1007/978-3-642-31057-7_10 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1871. Moshé M. Zloof & S. Peter de Jong (1977): The system for business automation (SBA): programming language. Communications of the ACM 20 (6). Pages 385-396. doi:10.1145/359605.359615 Exclusion reasons: Q5 [III.ajk]This article does not aspire to empiricity.

1872. S.H. Zweben, S.H. Edwards, B.W. Weide & J.E. Hollingsworth (1995): The effects of layering and encapsulation on software development cost and quality. Software Engineering, IEEE Transactions on 21 (3). Pages 200-208. doi:10.1109/32.372147 Exclusion reasons: Q1–2 [III.ajk]There isn't a clear language design decision at play.

1873. Ferad Zyulkyarov, Vladimir Gajinov, Osman S. Unsal, Adrián Cristal, Eduard Ayguadé, Tim Harris & Mateo Valero (2009): Atomic quake: using transactional memory in an interactive multiplayer game server. In Proceedings of the 14th ACM SIGPLAN symposium on Principles and practice of parallel programming. New York, NY, USA: ACM. PPoPP '09. Pages 25-34. doi:10.1145/1504176.1504183 Exclusion reasons: Q5 [III.ajk]This article is an extended experience report with no aspiration to empiricity save for some performance measurements that have no bearing on language design decision efficacy.

1874. Ferad Zyulkyarov, Srdjan Stipic, Tim Harris, Osman S. Unsal, Adrián Cristal, Ibrahim Hur & Mateo Valero (2010): Discovering and understanding performance bottlenecks in transactional applications. In Proceedings of the 19th international conference on Parallel architectures and compilation techniques. New York, NY, USA: ACM. PACT '10. Pages 285-294. doi:10.1145/1854273.1854311 Exclusion reasons: Q1–2 [II.ajk]This article deals with profiling, not with language design issue evaluation.

1875. I. T. B. Ørstavik (2008): Language shaping power: Bakhtin, Cassirer and Phenomenology. In XIIIth Bakhtin Conference. http://wiki.aitel.hist.no/the_art_of_programmers/images/archive/f/fd/20080801192307!Morphology_and_power.doc Exclusion reasons: Q1–2 [III.ajk]This paper takes a particularly ilnguistic approach to studying programming languages. It does not appear to engage in a comparative evaluation relevant to language design.

1876. 王晉華 (2009): 《C程序設計》教學中的開放式實驗初探. 電腦知識與技術 5 (34). Pages 9769-9770. http://d.wanfangdata.com.cn/Periodical_dnzsyjs-itrzyksb200934055.aspx Exclusion reasons: Q4 [II.ajk]Reported in what appears to be Chinese, and a Google Scholar search (published since 2009, keywords "jin-hua wang programming language" without the quotes) did not report anything similar enough in other languages.

APPENDIX 6 THE REJECTED DATA EXTRACTION FORM

The following data extraction form was initially planned for this study and was used in the pilot extraction. The form was accompanied with instructions, the Vessey, Ramesh, et al. (2005) taxonomy of research methods, and the Definitions 1 (on page 14), 2 (on page 14), 3 (on page 19), and 4 (on page 53). The form was found unsuitable and replaced by other methods.

	Your name	
	Date	

STUDY IDENTIFICATION

1	Study ID	
2	Authors of the paper(s)	
3	Title(s) of the paper(s)	

STUDY TYPE

Please concentrate only on the empirical content of the study. Many studies also discuss non-empirical matters.

4	Is this a primary or a secondary study?	
5	What programming language design decisions are studied in this study?	
6	How is the efficacy of programming language design decisions measured in this study?	

FOR PRIMARY STUDIES

7p	List all empirical research methods that apply from the Vessey et al taxonomy.	
8p	Describe any other empirical research methods used in this study.	

FOR SECONDARY STUDIES

7s	Does this study claim to be a systematic literature review or a systematic mapping study?	
8s	Does this study actually use systematic review methodology?	