JYVÄSKYLÄ STUDIES IN COMPUTING
222

Antti-Juhani Kaijanaho

Evidence-Based Programming Language Design

A Philosophical and Methodological Exploration

JYVÄSKYLÄN YLIOPISTO

JYVÄSKYLÄ STUDIES IN COMPUTING 222

Antti-Juhani Kaijanaho

Evidence-Based Programming Language Design

A Philosophical and Methodological Exploration

Esitetään Jyväskylän yliopiston informaatioteknologian tiedekunnan suostumuksella
julkisesti tarkastettavaksi yliopiston vanhassa juhlasalissa S212
joulukuun 4. päivänä 2015 kello 12.

Academic dissertation to be publicly discussed, by permission of
the Faculty of Information Technology of the University of Jyväskylä,
in building Seminarium, auditorium S212, on December 4, 2015 at 12 o'clock noon.

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2015

Evidence-Based Programming Language Design

A Philosophical and Methodological Exploration

JYVÄSKYLÄ STUDIES IN COMPUTING 222

Antti-Juhani Kaijanaho

Evidence-Based Programming Language Design

A Philosophical and Methodological Exploration

UNIVERSITY OF JYVÄSKYLÄ

JYVÄSKYLÄ 2015

Editors
Timo Männikkö
Department of Mathematical Information Technology, University of Jyväskylä
Pekka Olsbo, Ville Korkiakangas
Publishing Unit, University Library of Jyväskylä

URN:ISBN:978-951-39-6388-0
ISBN 978-951-39-6388-0 (PDF)

ISBN 978-951-39-6387-3 (nid.)
ISSN 1456-5390

Copyright © 2015, by Antti-Juhani Kaijanaho and University of Jyväskylä

Jyväskylä University Printing House, Jyväskylä 2015

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)

The most elementary and valuable statement in science, the beginning of wisdom, is "I do not know".
— Data in the Star Trek The Next Generation episode "Where Silence Has Lease"

Human beings, he among you is wisest who knows like Socrates that he is actually worthless with respect to wisdom.
— attributed to the god in Delphi by Socrates (Plato 2014, at 23b).

ABSTRACT

Kaijanaho, Antti-Juhani
Evidence-Based Programming Language Design. A Philosophical and Methodological Exploration.
Jyväskylä: University of Jyväskylä, 2015, 256 p.
(Jyväskylä Studies in Computing
ISSN 1456-5390; 222)
ISBN 978-951-39-6387-3 (nid.)
ISBN 978-951-39-6388-0 (PDF)
Finnish summary
Diss.

Background: Programming language design is not usually informed by empirical studies. In other fields similar problems have inspired an *evidence-based* paradigm of practice. Such a paradigm is practically inevitable in language design, as well. *Aims:* The content of evidence-based programming design (EB-PLD) is explored, as is the concept of evidence in general. Additionally, the extent of evidence potentially useful for EB-PLD is mapped, and the appropriateness of Cohen's kappa for evaluating coder agreement in a secondary study is evaluated. *Method:* Philosophical analysis and explication are used to clarify the unclear. A systematic mapping study was conducted to map out the existing body of evidence. *Results:* Evidence is a report of observations that affects the strength of an argument. There is some but not much evidence. EB-PLD is a five-step process for resolving uncertainty about design problems. Cohen's kappa is inappropriate for coder agreement evaluation in systematic secondary studies. *Conclusions:* Coder agreement evaluation should use Scott's pi, Fleiss' kappa, or Krippendorff's alpha. EB-PLD is worthy of further research, although its usefulness was out of scope here.

Keywords: programming languages, programming language design, evidence-based paradigm, philosophical analysis, evidence, systematic mapping study, coder agreement analysis

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

Reviewers

Professor Matthias Felleisen
College of Computer and Information Science
Northeastern University, USA

Professor Andreas Stefik
Department of Computer Science
University of Nevada, Las Vegas, USA

Opponent

Professor Lutz Prechelt
Institute of Computer Science
Freie Universität Berlin, Germany

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.

I am honored to have had Professor Matthias Felleisen of the Northeastern University and Professor Andreas Stefik of the University of Nevada, Las Vegas, as the reviewers of this dissertation and to have Professor Lutz Prechelt of the Freie Universität Berlin as the opponent in the defense. The valuable comments of the reviewers, particularly the extensive comments by Professor Stefik, have resulted in minor changes to the manuscript prior to publication.

This dissertation extends my Licentiate Thesis (published as Kaijanaho 2014), which was examined by Doctor Stefan Hanenberg of the University of Duisburg–Essen and Professor Stein Krogdahl of the University of Oslo. My thanks to them for their valuable 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 more than 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.

A significant part of this dissertation was written while I was visiting the Chair of Data Management Systems and Knowledge Representation at the University of Duisburg–Essen in Essen, Germany, from the beginning of February to the end of April 2015. My thanks to Professor Rainer Unland and Doctor Stefan Hanenberg for making this visit possible at their end; and to the two local doctoral students—Mr. Hans-Theodor Hünewinkel and Ms. Zohreh Akbari—who shared their office with me during the visit. A thank-you is also due to Professor Tuomo Rossi and the Department of Mathematical Information Technology, who suggested and funded the trip, respectively.

While in Essen, I had many deep conversations with Dr. Hanenberg on empirical studies and the philosophy of science, among other topics. We hold very different views on the philosophy of science and in particular on the manner in which computing research should be conducted, although we agree on the need for empirical human-factors research; this made the discussions very interesting to me and, I believe, to Dr. Hanenberg as well. My presentation of the ideas in this dissertation were influenced by those discussions.

I participated in the Dagstuhl Seminar 15222 “Human-Centric Development of Software Tools” at the end of May 2015 as this dissertation was in the final stages of preparation. I wish to thank the organizers and other participants of the seminar for enlightening discussions which (among other things) illuminated my thinking regarding some of the last difficult issues in this dissertation and thus helped me make this dissertation better. I wish to especially thank the participants of the breakout session on the philosophy of science.

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 my Licentiate Thesis acknowledgements, I also thanked the participants of the 15th International Conference on Evaluation and Assessment in Software Engineering (EASE) at Durham University, England in 2011; three anonymous reviewers of EASE 2012; and the interlibrary loan service of the university library. These thanks apply also to the relevant parts of this dissertation, but for the details, please see the Licentiate Thesis acknowledgements.

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

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

LIST OF FORMAL NOTATION USED IN CHAPTER 4

Notation used in the metalanguage only

$\mathbb{N}$	the set of all nonnegative integers $\{0, 1, 2, \dots\}$
$\mathbb{R}$	the set of real numbers
$[0,1]$	$\{\, r \in \mathbb{R} \mid 0 \leq r \leq 1 \,\}$
$]-1,1]$	$\{\, r \in \mathbb{R} \mid -1 < r \leq 1 \,\}$
$\mathcal{P}(S)$	powerset of S
$\mathcal{L}$	an arbitrary finitary quantifier-free first-order predicate language
$\mathcal{L}^1$	an extension of $\mathcal{L}$ to allow finitary universal and existential quantifiers
$\mathcal{L}_\omega$	an extension of $\mathcal{L}$ to allow countably infinite conjunctions and disjunctions
$\mathcal{L}^1_\omega$	an extension of $\mathcal{L}$ to allow finitary universal and existential quantifiers and countably infinite conjunctions and disjunctions
p, q	arbitrary formulas or sentences of the language under discussion
K	an arbitrary set of sentences of the language under discussion, usually used as (a part of) a premise of an argument
$\mathcal{H}$	the set of all variable-free terms of the language under discussion
$\mathcal{S}$	the set of all sentences of the language under discussion
$p[t/x]$	the formula obtained by substituting the term t for all free occurrences of the variable x within p, taking care to avoid variable capture
$\mathbb{M}$	the set of all interpretations of the language under discussion
$\mathcal{M}$	an arbitrary interpretation of the language under discussion
φ	a variable assignment: a mapping from variable symbols to individuals of an interpretation
$[\![t]\!]^\varphi_{\mathcal{M}}$	the individual of $\mathcal{M}$ that the term t denotes given the variable assignment φ (if t contains no variables, φ may be omitted)
$\mathcal{M} \vDash_\varphi p$	the formula p is true in the interpretation $\mathcal{M}$ given the variable assignment φ (if p is a sentence, φ may be omitted)
$\mathcal{M} \nvDash_\varphi p$	the formula p is false in the interpretation $\mathcal{M}$ given the variable assignment φ (if p is a sentence, φ may be omitted)
$p^{\mathcal{M}}$	p if $\mathcal{M} \vDash p$ and $\neg p$ otherwise
$[\![A]\!]$	the strength of the admissible argument A
$\mathcal{A}$	the set of all admissible arguments
$(V, \leq)$	the partially ordered set from which strength values are picked
V^*	$\{\, v \in V \mid \exists A \in \mathcal{A}\colon\ [\![A]\!] = V \,\}$
$\top$	the strength assigned to all valid admissible arguments
$\bot$	the strength assigned to all self-contradictory admissible arguments

C	a function $V \times V \to V$ used to define the strength of an argument whose conclusion is a conjunction
D	a function $V \times V \to V$ used to define the strength of an argument whose conclusion is a disjunction
$\mathcal{T}$	a function mapping a sentence to the set of interpretations in which it is true
$\mathcal{T}_K$	$\mathcal{T}$ restricted to consider only interpretations where the sentences in K are true
$\operatorname{ran} f$	$\{ y \mid \exists x \colon y = f(x) \}$ for any function f
$\Rightarrow$	material conditional
$\Leftrightarrow$	material biconditional
$P(A)$	the probability of the set A (the denotation of P varies)
$P_K(A)$	the probability of the set A given the sentences in K (the denotation of P varies)
$P(A \mid B)$	$\dfrac{P(A \cap B)}{P(B)}$ (also with a subscript K)
$P(p)$	$P(\mathcal{T}(p))$ (also with a subscript K)
$P(p \mid q)$	$P(\mathcal{T}(p) \mid \mathcal{T}(q))$ (also with a subscript K)
$K \nvdash p$	the argument $K \vdash p$ is self-contradictory

Notation used in the metalanguage and the formal languages

$\neg p$	the logical negation of formula p
$p \wedge q$	the logical conjunction of formula p and q
$p \vee q$	the logical disjunction of formula p and q
$\bigwedge Q$	the conjunction of a countable set of formulas Q (variants are also in use)
$\bigvee Q$	the disjunction of a countable set of formulas Q (variants are also in use)
$\forall x p$	the universal quantification of p with respect to x
$\exists x p$	the existential quantification of p with respect to x

Notation used in the formal languages only

$F(t_1, \ldots, t_n)$	application of function symbol F to terms $t_1, \ldots, t_n$
$R(t_1, \ldots, t_n)$	application of predicate symbol R to terms $t_1, \ldots, t_n$
$K \vdash p$	an argument with the sentences K as premises and the sentence q as the conclusion
$p \to q$	material implication in the languages under discussion (defined as $\neg p \vee q$)
$\mathsf{A}p$	modal sentence for the acceptance of p
$\mathsf{R}p$	modal sentence for the rejection of p

LIST OF FIGURES

LIST OF TABLES

CONTENTS

1 INTRODUCTION

The genesis of this dissertation is a question I asked myself over a decade ago. How can I, as an aspiring programming language designer, improve upon existing languages instead of simply making another variation of a well understood theme? The well established research paradigm in language design would have me identify a deficiency in the expressiveness[1] of some class of languages and then propose an alteration of these languages which arguably removes the deficiency. However, there already is several decades' worth of such contributions, suggesting two much more pressing questions. How does one compare design alternatives that are present in the literature? And the question I focus on in this dissertation: *How does one use the existing literature to aid in language design?*

In this dissertation, my primary focus is on what I would like to call multi-decennial language design—the slow advance of general-purpose programming languages in actual mission-critical use, both the evolution of existing languages and the occasional slow-motion revolutions in which languages are replaced by new ones. In this language design environment, individual design choices may have huge economic consequences, and it seems appropriate to make such choices with all due deliberation. The use of existing literature is also of special interest to language designers working on educational languages, that is, languages used in teaching programming. However, the ideas I discuss in this dissertation are certainly applicable to the fast-paced design of domain-specific languages as well (at least to the extent that they are expressive enough to qualify as programming languages). However, research languages, which are mainly vehicles for expanding the state of the art in language design, are not my concern here.

The issue of using existing literature to guide practical decisions is relevant in many other disciplines. Clinical faculty at the McMaster University developed in the 1970s an approach to reading medical journals for busy medical practition-

[1] I mean here something more fine-grained than simply the set of number-theoretic functions that can be defined in it, which is what Turing-completeness is about. The expressibility theories of, e. g., Felleisen (1991), Mitchell (1993), and Shapiro (1991) appear good *prima facie* candidates for explicatums of this concept, but I do not need to examine this question in detail here.

ers (Smith and Rennie 2014) and published a series of guides on it in the early 1980s (e. g., Sackett 1981). During the following decade, they developed a complete approach to using the medical literature to guide an individual physician to come to a decision as to how to handle a particular patient's medical problem. They called this approach *evidence-based medicine* (Evidence-Based Medicine Working Group 1992); it has become a very influential movement in medicine (see, e. g., Hitt 2001; Greenhalgh et al. 2014; Godlee 2014) and spawned derivative movements in other disciplines such as management (Briner et al. 2009), policing (Sherman 1998), nature conservation (Pullin and Knight 2001), policy-making (Nutley et al. 2010), and software engineering (Kitchenham, Dybå, et al. 2004).

No such derivative currently exists in the field of programming language design, as I argue in Section 10.1. In this dissertation, I examine the following question:

RQ1: What should evidence-based programming language design (EB-PLD) consist of?

Given that EB-PLD does not exist at this point, there is nothing for empirical research methods to do respecting this question. Nor is a mathematical approach viable, as I would first have to set up an axiom system from which theorems could be derived, and the process of setting up that axiom system would all but answer my research question. At this point, some authors (cf., e. g., Hume 1748/2011, last paragraph; Wittgenstein 1921/1974, Prop. 6.53; Ayer 1936/2001)[2] would counsel me to drop this question. However, there is in my view value in clarifying the concept of EB-PLD for future use. The name itself suggests a vague notion, and anyone familiar with evidence-based medicine can readily imagine the general contours of what a corresponding concept for programming language design might be, but a detailed analysis or explication of the concept is missing. For clarifying unclear ideas, a suitable method is philosophical analysis and explication, discussed in Chapter 3.

In short, I will argue in Chapter 9 that the Evidence-Based _____ concept is an idiom derived from Evidence-Based Medicine; that its core is a five-step process performed by an individual practitioner to resolve, using the research literature, uncertainty about how to proceed with an actual problem. Therefore, as I argue in Chapter 10, Evidence-Based Programming Language Design ought to be an analoguous five-step model used by actual language designers to use the research literature to resolve actual language design problems. I hasten to add, however, that Evidence-Based Programming Language Design is intended to coexist with other approaches to language design, to be called in only when there is actual uncertainty.

In support of the hypothetical Evidence-Based Programming Language approach, I posed myself the following question:

[2] Hume (1748/2011, last paragraph, emphasis in the original): "If we take in our hand any volume; of divinity or school metaphysics, for instance; let us ask, *Does it contain any abstract reasoning concerning quantity or number?* No. *Does it contain any experimental reasoning concerning matter of fact and existence?* No. Commit it then to the flames: for it can contain nothing but sophistry and illusion."

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

I answered it in my Licentiate Thesis (Kaijanaho 2014), which is incorporated in this dissertation largely verbatim;[3] I will note separately each time the Licentiate Thesis is being reused. This question is empirical in nature and I therefore answered it using the best-practice methodology, that is, as a systematic mapping study (reviewed generally in Chapter 5). The mapping study, unlike the rest of this dissertation, is limited to textual programming languages. I will discuss the research design of the mapping study in Chapter 7 and its results in Chapter 8.

The mapping study required me to explicate the concept of scientific evidence; this need was pointed out to me by Dr. Vesa Lappalainen in personal communication. I based my explication in the Licentiate Thesis on a rather shallow understanding of Bayesian epistemology. In this dissertation, I have reexamined the issue. In Chapter 4, I have explored a foundational theory of induction, inspired by Bayesian epistemology, which then suggests a rather natural explication of evidence. In that chapter, I therefore answer the following question—

RQ2: What does the word "evidence" mean?

—and relate my new answer to the explication used in the mapping study.

The mapping study further required me to perform various coder agreement assessment exercises, as discussed in Chapter 7. Their purpose is to evaluate the reliability of human-generated coded data by seeing how similar results do different people working independently generate from the same initial input. It is standard to recommend the use of Cohen's κ for summarizing the result of such an exercise (see, e. g., Kitchenham and Charters 2007) but such recommedations do not usually come with an argument or citations to literature to demonstate the appropriateness of such a recommendation. Digging into the literature myself, I noticed that some methodologists do dispute this recommendation (see, e. g., Krippendorff 2004; Gwet 2012), suggesting alternatives. This was a cause for concern, and therefore I asked the following question:

RQ3: Is Cohen's κ appropriate for assessing coder agreement in a systematic secondary study, and if not, what replacement should be used?

To answer this question, I review in Chapter 6 the methodological literature and the mathematical foundations of Cohen's κ and several proposed alternatives. My conclusion necessitates a limited reassessment of the Licentiate Thesis agreement analyses.

In the end, in this dissertation I will be arguing for a specific understanding of what "evidence-based" means in the context of programming language design. Mathematically trained readers may object, arguing that definitions are arbitrary; but definitions are rarely (even in mathematics) truly arbitrary, as they are typically either formalizations of real-world concepts or abstractions based on

[3] It is a well established tradition in Finnish post-graduate studies that a Licentiate Thesis, if any, is used as a part of a doctoral dissertation, though it is not mandatory to do so.

existing mathematical concepts, and in both cases sharpened by proof attempts and their fixes (see, e. g., Lakatos 1976). The "evidence-based" concept is already well known, and it is practically inevitable that the phrase will end up used in the context of programming language design, and the phrase is rather obviously value-laden; thus, its correct definition matters.

However, my explication of Evidence-Based Programming Language Design (EB-PLD) occupies a fairly small number of pages of this dissertation. This is because EB-PLD is the tip of an iceberg. Just below the surface is the body of science on which EB-PLD would draw; further below the surface lies the methodology and the philosophy of science. I could have simply stayed above the surface, but that would have given a rather limited view of the actual situation. Instead, I have charted a much larger part of the iceberg, somewhat to the detriment of detail in the map of the tip.

This dissertation is in essence a proposal for a Lakatosian research program. Lakatos (1970/1978) saw science as advancing through research programs, each of which consists of a series of theories; in this he attempted to marry the rather different theories of scientific progress offered by Popper (1935/2002) and Kuhn (1962/1996). Each successive theory should improve on the previous by making new predictions, and once in a while (but not all the time) a progressive research program will see its predictions be empirically validated; if a program does not progress empirically, it will eventually die. In this dissertation, I outline what could be viewed as a theoretical framework for Evidence-Based Programming Language Design, from which several empirical predictions can be derived. I leave the empirical evaluation of those predictions (and the all but inevitable correction of my framework in response of such evaluation) for later work.

If not by empirical results, how should this dissertation be appraised? I will admit at the outset that I do not expect all my readers to agree with my conclusions; compelling agreement is only possible to mathematical proof, which is inapplicable here. I will concede that self-contradictory arguments (such as serious errors of mathematics) are below par. Beyond that, my goal is to convince all my readers that my position is reasonable, even though some of them may end up disagreeing with me on the substance. For those readers, I intend to give a clear enough position and a clear enough argument that they can develop a clear counter-argument, so that the point of disagreement can be debated in the best tradition of scholarship.

I have written this dissertation using the singular first person. It is utterly standard in philosophical writing (see, e. g., Hyland 2001b; Sword 2012); as an anonymous philosopher told Hyland (2001b, p. 217) in an interview:

> "The personal pronoun 'I' is very important in philosophy. It not only tells people that it is your own point of view, but that you believe what you are saying. It shows your colleagues where you stand in relation to the issues and in relation to where they stand on them. It marks out the differences."

When writing a philosophical argument, I am not promulgating laws of nature but examining a position on which reasonable people can (usually) disagree. Thus, when I assert something, it is merely I who is asserting it; I will not claim

the royal "we" that refers to all scientists as a class, nor will I arrogantly assume that you agree with me by saying that "we", that is, you and I, assert something. Using short passive constructions (though liguistically legitimate, see Pullum 2014) could signal that I do not hold the position myself, which should make the reader wonder why; alternatively, it can be seen as a disguised version of the royal "we". The use of the singular first person is further appropriate when reporting a mapping study; after all, in a systematic secondary study, establishing the audit trail—who did what—is an essential part of the methodology (for precedent, see, e. g., Kitchenham 2010).

I will, in addition, occasionally use rhetorical techniques like addressing the reader directly; that is, I will sometimes mention you (yes, you) explicitly. As I mentioned above, my task in this dissertation is to attempt to convince you, and in certain special circumstances (particularly when I am meta-discussing an argument I have given) it will be appropriate to be explicitly reflective about that aspect of this dissertation. Hyland (2001a) and Sword (2012) have shown that this usage is not unknown in scholarly writing, particularly in philosophy.

I will briefly mention two other conventions I use in this dissertation. First, 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. Second, when referring to certain literature that I have access to as reprints, translations, or scholarly editions (e. g., Wittgenstein 1921/1974), I first give an approximate year of original publication and only then the year by which the source is listed in my bibliography, in order to make clear the relative age of my various sources.

This dissertation contributes to the field of programming language design in the following manners:

- The extent of existing empirical evidence on efficacy of language design decisions is shown.
- The sense of evidence-based programming language design is explicated.

In addition, this dissertation contributes to the methodological literature in programming languages and software engineering as follows:

- A philosophical analysis approach for the clarification of unclear concepts is exposited.
- A theory of inductive reasoning is exposited, leading to a theory of empirical evidence.
- Commonly advocated statistics for assessing coder reliability in systematic secondary studies are theoretically evaluated.

In all three cases, the ideas are largely derived from existing and well known bodies of research in their respective fields (philosophy, for the first two, and psychometrics and content analysis for the third) and would not, I think, be considered significant advances of the literature in their home fields, although there are probably minor contributions even to them. However, so far as I can tell, they

are largely unknown in the programming language and software engineering methodological literature.

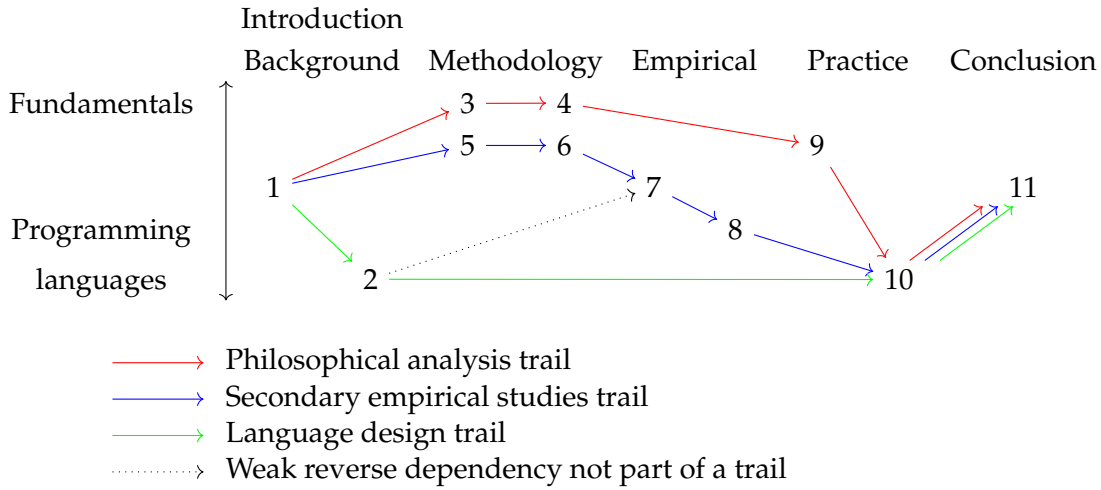

FIGURE 1 An outline of this dissertation. Horizontally, the chapters are arranged in thematic order, and vertically, they are (very roughly) sorted according to their direct relevance to language designers. Several trails through this dissertation are displayed that skip some chapters; though reading all chapters in numeric order is preferred. All arrows, no matter their visual appearance, further indicate a weak reverse dependency, suggesting that reading the source of the arrow before its target may be helpful.

Figure 1 gives an outline of this dissertation. The chapters are intended to be read in the order they are presented, and the dissertation is intended to be read from Introduction to Conclusion without skipping chapters. However, the figure shows three trails through the dissertation that allow readers who are so inclined to focus on specific lines of argument and skip some chapters; these trails also illustrate that there are more than one way I could have structured 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). This chapter appeared, with minor differences, in my Licentiate Thesis (Kaijanaho 2014).

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. Gabbrielli 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.

In the mapping study reported later in this dissertation, I have adopted 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. For the rest of this dissertation, either conception would be valid.

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) analyzes programming in terms of what appears to make it difficult. The act of programming is, according to his analysis, 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

[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.

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 language), and often deliberately uses abstraction to manage complexity. This analysis is not derived from or backed by direct empirical evidence, but it appears highly plausible.

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 Analysis

Combining all these threads yields an analysis of Programming Language. In this dissertation, I will be further requiring 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.[2] This yields the following analysis:

Analysis 1. *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*[3] *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] In their evaluation report of my Licentiate Thesis, Hanenberg and Krogdahl suggest that it would have been simpler to just use "the common phrase 'general purpose programming language'" (p. 1) here. I do not think that phrase means the same as my analysis below; and in any case, the term is too vague to apply in the edge cases.

[3] 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 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 the mapping study reported later in this dissertation.

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 usually 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 (1962/1996, 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,[4] 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

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

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 my mapping study employ them. The following programming paradigms are of special interest to 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 analyzed a programming language (Analysis 1 on page 24) 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 dissertation.

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.[5]

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

[5] Strictly speaking, abstract syntax is truly abstract and does not involve actual trees. For the purposes of this dissertation, 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 (1962/1996) in 1962; an elegant mathematical formulation based on universal algebra was given by Gougen et al. (1977).

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 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 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 dissertation.

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 the 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 the 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

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

when the value of a is 1?[6] A number of solutions were proposed in the years following the publication of ALGOL 60 (see Abrahams 1966), including revising 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 (van 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:

```
x := 0;                          x := 0;
if a > 0 then                    if a > 0 then
   if a > 2 then                    if a > 2 then
      x := 1                           x := 1
   end a > 2                        not a > 2 then
not a > 0 then                         x := 2
   x := 2                           end a > 2
end a > 0                        end a > 0
```

Embley and Hansen (1976) and Embley (1978) defined a new control structure unifying iteration and conditionals, the *KAIL selector*. The following is an

[6] 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).

example given by Embley (1978, p. 200, direct quote):

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

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)[7] 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.[8] 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[9] 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 ap-

---

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

8 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.

9 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.

plied 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 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,[10] Python,[11] JavaScript (see, e. g., Mikkonen and Taivalsaari 2008), and Ruby.[12]

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

10 http://www.perl.org/
11 https://www.python.org/
12 https://www.ruby-lang.org/

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.

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.[13] 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

[13] As Domenico Ruoppo pointed out to me in private correspondence in June 2015, Dana Scott (1969/1993) roughly contemporaneously suggested the use of typed lambda calculus as a foundation for the theory of computation. That long unpublished and privately circulated note inspired the automated reasoning system that gave birth to the ML language (see, e. g., Gordon et al. 1979); it was not, however, itself a programming language paper.

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.[14]

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,[15] 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,

[14] After writing this subsection in Spring 2014 and publishing it as part of my licentiate thesis (Kaijanaho 2014), I have become aware of a roughly contemporaneously written and published article by Kell (2014), which explores largely the same issues in a slightly different manner. Our basic conclusion is the same: there are two traditions of types—one originating from practical programming language design issues and the other originating from the early 20th Century research in logic—which still are not consolidated to a unified concept of type.

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

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 dissertation 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* 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 the 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". 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 it generally unlike that usually expected of an empirical discipline.

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, in Détienne's assessment, 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 8. There is, however, a line of research not included in my 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.[16] 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[17], whose design was influenced by and tested in several published studies of programmer behavior (at least Mayer et al. 2012b; Stefik and Siebert 2013). Third, the textbook of Klerer (1991) discussed the use of programmer behavior research to inform language design.

---

16 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.

17 http://quorumlanguage.com/

The main thrust of this dissertation is to explore the possibility that this situation might be changed. First, however, I need to discuss some foundational issues.

3 PHILOSOPHICAL ANALYSIS

The principal research approach of this dissertation is philosophical analysis. In this chapter, I will describe those traditions of analysis in philosophy that I draw upon, to give you an understanding of what I mean when I say that I analyze or explicate a concept. I will also argue that philosophical analysis is a—not very common but—recognized approach in computing and programming language research.

3.1 Analysis in philosophy

In this section, I will be arguing for a specific view of analysis as it is practiced by philosophers. The argument consists mainly of examples drawn from well known philosophers, along with an explanatory narrative; an additional premise I will be using without further mention is this: "if a philosopher has practiced philosophy in a particular way, it is a philosophical practice". I make no claim that the view I present is exhaustive or exclusive of any other view; I do not need to, as my only goal in this section is to explain the kind of analysis I will be pursuing in this dissertation.

In short, the view of analysis presented below is this: analysis is, on the one hand, the clarification of unclear concepts by giving a specific multi-part elaboration (called an *analysis*) of it, and on the other hand, the reasoned argument in support or against a specific such analysis. I will argue that a philosophical argument is similar to a mathematical one, and I will discuss several important differences between them.

Further, I will be arguing, by reference to actual philosophical practice on the one hand and to textbook descriptions of it on the other, that philosophical arguments share the same basic structure as other logical arguments—there are premises and conclusions as well as a warrant for the inference—but that deductive reasoning is not the only kind of reasoning used. I will also examine certain ways premises are justified short of another argument.

Additionally, I will be arguing that analysis is usually not unassailable and that analyses are frequently contested by other philosophers.

3.1.1 The nature of philosophy

It is much harder to *explain* philosophy than it is to *do* philosophy. For any possible characterization of it that I could write down, some philosopher somewhere, either in the past, in the present, or in the future, is going to disagree with it. "After all," like Plant (2012, fn. 13 on p. 571) writes, "genuinely uncontroversial claims in philosophy tend to be trivial and uninteresting." As an extreme example to illustrate my point, consider that there have been philosophers who maintained that philosophy itself (including their own) is nonsense. Ludwig Wittgenstein ended his famous *Tractatus Logico-Philosophicus* (1921/1974) in these words (emphasis in the original):

> "6.53 The correct method in philosophy would really be the following: to say nothing except what can be said, i.e. propositions of natural science—i.e. something that has nothing to do with philosophy—and then, whenever someone else wanted to say something metaphysical, to demonstrate to him that he had failed to give a meaning to certain signs in his propositions. Although it would be unsatisfying to the other person—he would not have the feeling that we were teaching him philosophy—*this* method would be the only strictly correct one.
>
> "6.54 My propositions serve as elucidations in the following way: anyone who understands me eventually recognizes them as nonsensical, when he has used them—as steps—to climb up beyond them. (He must, so to speak, throw away the ladder after he has climbed up it.)
>
> "He must transcend these propositions, and then he will see the world aright.
>
> "7 What we cannot speak about we must pass over in silence."

It should surprise nobody that I disagree with the young Wittgenstein on this topic, for otherwise I would not have written this dissertation in this particular manner. However, it ought to be understood that the following description of philosophical analysis is by necessity incomplete, or worse.

I begin with Wittgenstein's own initial analysis of Philosophy[1] in the beginning of the *Tractatus* (from Proposition 4.112):

> "Philosophy aims at the logical clarification of thoughts. Philosophy is not a body of doctrine but an activity. [...] Without philosophy thoughts are, as it were, cloudy and indistinct: its task is to make them clear and to give them sharp boundaries."

Of course, Wittgenstein gave very specific technical definitions for some of the terms in the analysis elsewhere in the *Tractatus*. I do not restrict myself to his definitions, but otherwise this is, more or less, the understanding of Philosophy that I apply in this dissertation.

A caveat is in order here. I hold that philosophy is necessarily prior to science, in that to practice science, one must adopt a philosophical position. From this position then follow positions on the appropriate scope of science and on methodology. This adoption may be (and probably usually is) implicit, which may explain that Hawking and Mlodinow (2010, p. 14) could cavalierly declare—

[1] I follow here Rudolf Carnap (1956, fn. 10 on p. 17, 1962, p. 8) convention of capitalizing the name of a concept when I need to specifically name it as an object of study to avoid potential confusion.

> "Traditionally these are questions for philosophy, but philosophy is dead. Philosophy has not kept up with modern developments in science, particularly physics."

—in a book that is, in my view, nothing but (physics-informed) philosophy (cf. Norris 2011). I thus agree with Blackmore (1979, fn. 1 on p. 125)—

> "By 'philosophy' I normally mean the study of our most basic universal assumptions or the assumptions themselves"

—but for the purposes of this dissertation, the Wittgenstein characterization is more useful.

3.1.2 Static structure of analysis

Analysis is a millennia old approach to going about the clarification of unclear ideas. I will be using Knowledge as a running example throughout this chapter. Plato analyzes it in Meno 98a (Plato 2005, p. 139, emphasis added)—

> "As long as they stay put, *true beliefs* too constitute a thing of beauty and do nothing but good. The problem is that they tend not to stay for long; they escape from the human soul and this reduces their value, unless they're *anchored by working out the reason*. [...] When true beliefs are anchored, they *become pieces of knowledge* and they become stable."

—and in Theaetetus 202c (Plato 1984, p. I.74, emphasis added)—

> "whenever anyone gets the *true opinion* of anything without speech, his soul tells the truth about it but does not know, for whoever is *incapable of giving and receiving an account* (speech) is *without knowledge*"

Both passages identify three qualities associated with Knowledge: (1) it is true, (2) it is a belief, or an opinion, and (3) a reason for (or an account of) it has been worked out. Gettier (1963) gives the same analysis the following modern phrasing:[2]

> "S knows that P IFF (i) P is true,
> (ii) S believes that P, and
> (iii) S is justified in believing that P."

A conventional tagline for this classical analysis, however phrased, is "Knowledge is Justified True Belief", often abbreviated "JTB" (see, e. g., Turri 2012).

At the most superficial level, analysis itself is conventionally analyzed as consisting of two things (see, e. g., Langford 1942/1968): the thing that is being analyzed, called the *analysandum*, and the complex of things that are offered as its analysis, called the *analysans*. For example, in my running example, the analysandum is Knowledge and the analysans was justified true belief.

Philosophers disagree about what sort of things the analysandum and the analysans can be. For example, Wittgenstein in the *Tractatus* (1921/1974) regarded them as being phrases in a language, starting the influential (but eventually mostly abandoned) linguistic turn (Rorty 1992; Hacker 2013); under this

[2] The word "IFF" is short for "if and only if" and conventionally indicates that the conditions are both necessary and sufficient.

view, JTB does not analyze Knowledge but the natural-language word "knowledge". A related view, called the conceptual turn by at least Butchvarov (2003) and Williamson (2007), regards the analysandum and the analysans as states of mind that adherents of this view call concepts; thus the analysandum of the JTB analysis would be the state of mind of a person possessing knowledge. However, when Plato discussed the JTB analysis, he did not intend to describe the Greek word for knowledge or the state of mind of a person possessing knowledge; what he tried to grasp was the true nature of knowledge, which exists independently of humans and can only be imperfectly grasped.

Philosophers also disagree on the relation that an analysis purports to establish between the analysandum and the analysans. One could posit that a successful analysis will establish that the analysandum and the analysans apply to exactly the same class of existents (which is traditionally phrased as the analysans being both necessary and sufficient for the analysandum). But this view gives rise to a problem, called the *paradox of analysis* by Langford (1942/1968) and many others: If the analysandum and analysans apply to the same existents, then they say the same thing and are interchangeable, and in particular, the analysis remains the same even if the analysans is replaced by the analysand, and thus the analysis is trivial; if they do not apply to the same existents, the analysis is incorrect. For example, if JTB is a correct analysis of Knowledge, then (by the argument in the paradox) justified true belief is interchangeable with Knowledge, and in particular, the JTB analysis can be rephrased as "Knowledge is necessary and sufficient for Knowledge", which is certainly a triviality.

One way to sidestep the paradox was advanced by Carnap (1962). Instead of having the analysans be necessary and sufficient for the analysandum, he would instead have the analysans be a similar but distinct replacement, a better new version if it were, of the analysandum. Carnap called this sort of analysis *explication*, and used the terms *explicandum* and *explicatum* instead of analysandum and analysans, respectively. The point of Carnapian explication is the clarification of a prescientific (or an earlier scientific) concept, resulting in an exact (current-)scientific concept, somewhat akin to what quantitative researchers do when they operationalize a concept for measurement. On page 7, Carnap lists four requirements for a valid explicatum (emphasis in the original):

> "(1) The explicatum is to be *similar to the explicandum* in such a way that, in most cases in which the explicandum has so far been used, the explicatum can be used; however, close similarity is not required, and considerable differences are permitted.
> "(2) The characterization of the explicatum, that is, the rules of its use [...], is to be given in an *exact* form, so as to introduce the explicatum into a well-connected system of scientific concepts.
> "(3) The eplicatum is to be a *fruitful* concept, that is, useful for the formulation of many [...] empirical laws [or] logical theorems[...].
> "(4) The explicatum should be as *simple* as possible; this means as simple as the more important requirements (1), (2), and (3) permit."

Explication, as Carnap defines it, is very common in science. For example, one might anachronistically regard what Isaac Newton did in his *Philosophiae Naturalis Principia Mathematica* (1726/2012) to Force—defining it as the cause of all

change in motion—as explication; as Jammer (1957/1999) explains in detail, it significantly deviates from the previous understandings of the concept, yet is similar to them, and it certainly was exact, fruitful, and simple, as Carnap requires. Carnap himself gives several other examples (1962, p. 4–15), including the explication of salt as sodium chloride and the explication of Truth given by Tarski (1944).

A simpler solution to the paradox of analysis is to note that what existents a concept applies to does not define a concept completely (see, e. g., Church 1946). This is the topic of the next subsection.

3.1.3 Sense and denotation, intension and extension

Frege (1892/1948) famously analyzed the meaning of a proper name as having two distinct aspects. Obviously, there is the existent which the name denotes, its *denotation*.[3] However, some proper names have no denotation; for example, Sherlock Holmes and Narnia are both proper names which have no denotation. One solution would be to regard names with no denotation as being gibberish, but that would seriously restrict the usefulness of language. Another solution—decreeing that fictional entities exist—has its own issues. There is also another problem: if denotation were all there is to a proper name's meaning, then the rather astonishing statement that "the morning star" and "the evening star" denote the same celestial object becomes a triviality, akin to $a = a$. Frege, therefore, noted that a name must have an additional aspect to its meaning, which he called the name's *sense* (*Sinn* in the original German). Thus, the morning star and the evening star have the same denotation but different senses; similarly, Sherlock Holmes and Narnia have a sense but no denotation. Frege also extended this analysis from proper names to many kinds of complex phrases in ordinary language.

The Fregean distinction between sense and denotation is reminiscent of the classical analysis of Concept. It appears in an embryonic form in the *Categories* of Aristotle (2014), in which individual existents would be grouped into species and genera and given various adornments. In the very influential Port-Royal Logic (Arnauld and Nicole 1683/1996, p. 39–40, emphasis in the original), there appeared the notion of "universal ideas", two "things" in which was "most important to distinguish clearly": "the *comprehension* of an idea" consisted of "the attributes that it contains in itself, and that cannot be removed without destroying the idea", and its "*extension*", which consisted of those existents that the general idea applies to and any general ideas subordinate to it. Eventually (see, e. g., Hamilton 1861, p. 289–290) these general ideas became *concepts*, each of which had an *intension* which was a bundle of attributes (which Arnauld and Nicole called comprehension), and an *extension*, which consisted of the existents (but

[3] The proper translation of Frege's German term *Bedeutung* is controversial; see Beaney (1997a, p. 36–46). I have adopted "denotation" despite the disadvantages Beaney notes because it is familiar to programming language theorists and is close enough to Frege's intention for the purposes of this dissertation.

not subordinate ideas) that the concept includes. Now, a concept usually has a name; the Fregean sense of that name corresponds to the intension of the concept named, and the Fregean denotation of that name corresponds to the extension.

The Aristotle–Port-Royal classical analysis of Concept should be contrasted with modern reanalyzes; I will discuss three. The first is that of Frege (1891/1997), which is only a minor modification; he considered a concept to be a class of existents, thus roughly corresponding to the extension of a classical concept, but that concept is in practice always denoted by some word or phrase in a language, and thus the sense of that word or phrase corresponds to the classical intension. Carnap (1956), in contrast, working in a formal language the details of which are not important here, considered a concept to be a property, which he defined as an equivalence class, under logical equivalence, of declarative sentences with one free variable. For Carnap, intension and extension were not associated with concepts but were defined for all types of expressions in his formal language; in the case of declarative sentences with one free variable, the intension was the concept associated with it, and the extension was the class of existents that satisfy it.

Wittgenstein in his posthumous *Philosophical Investigations* (1953/2009) offered a much more radical reanalysis (§ 66–67, emphasis in the original):

> "Consider, for example, the activities that we call 'games'. I mean board-games, card-games, ball-games, athletic games, and so on. What is common to them all? — Don't say: 'They *must* have something in common, or they would not be called "games"' — but *look and see* wheter there is anything common to all. — For if you look at them, you won't see something that is common to *all* [...] we see a complicated network of similarities overlapping and cris-crossing: similarities in the large and small. I can think of no better expression to characterize these similarities than 'family resemblances' ; for the various resemblances between members of a family [...] overlap and cris-cross in the same way. — And I shall say: 'games' form a family."

Wittgenstein had taken the linguistic turn (Rorty 1992; Hacker 2013) and thus specifically analyzed the use of language (what he called "language-games") but the argument applies in full force to nonlinguistic concepts, as well: some, like Game, cannot be viewed as having a global intension consisting of things that are common to all games and (in concert) to nothing but games; instead, the concept is held together by local *family resemblances*.

The classical idea of a concept's intension as a collection of commonalities cannot survive Wittgenstein's argument, at least not in the general case (one may suppose that some concepts are simple enough that they have a classical intension). Yet, I believe there is work to be done by something that goes beyond a concept's extension: like it is necessary to distinguish between linguistic expressions denoting the same existent using their senses, it will be necessary to distinguish between coextensive concepts (that is, concepts that have the same extension). I will henceforth call whatever distinguishes a concept from all other coextensive concepts its *intension*. It may be the classical list of commonalities or it may be something else; I do not need to offer an analysis of it in this dissertation.

Now, the simple solution of the paradox of analysis can be specified with exactness: an analysis may claim the coextensionality of the analysandum and the analysans while being safe from the paradox so long as their intensions differ.

3.1.4 Philosophical argument

The aspects of analysis I have described above all relate to what might be called a static view of it; in the Wittgensteinian (1921/1974, Prop. 4.112) terms I began this section with, an analysis has so far been viewed as "a body of doctrine". However, "[p]hilosophy is [...] an activity." In particular, it is a social activity, with philosophers publishing their views and other philosophers then conducting critical reanalyses of them; those reanalyses are published, and the cycle starts again. It is very common for philosophers to disagree, even to the point that it is doubtful whether there is any progress in philosophy (see, e. g., Dietrich 2011).

The publications of philosophers, to the extent they offer analyses, contain not just the static-view analysis of the sort I described above but often also arguments designed to persuade the reader of the analysis' correctness. Some cases where no argument is offered—such as parts of Wittgenstein's *Tractatus* (1921/1974)—may perhaps be best viewed as presenting material that is (in the eyes of the author, and of everyone who learns from it) *retroactively obvious*,[4] that is, ideas that one had not imagined beforehand but once seen become so obvious that one nearly forgets ever having been ignorant of them. Such ideas are likely indeed best served by being presented without argument.

Walton (1990, p. 411) defines an argument as "a social and verbal means of trying to resolve, or at least to contend with, a conflict or difference [...] between two (or more) parties", which could be "an unsolved problem [or] an unproven hypothesis". In the context of philosophical analysis, the parties are typically the philosopher and their audience, with the former advancing a claim and the latter adopting a skeptical attitude, waiting to be convinced. Within an argument, *reasoning* is used in order to convince or refute. Walton defines it as follows (p. 403, emphasis in the original):

> "*Reasoning* is the making or granting of assumptions called *premises* (starting points) and the process of moving toward conclusions (end points) from these assumptions by means of warrants. A *warrant* is a rule or frame that allows the move from one point to the next point in the sequence of reasoning."

It is very common not to distinguish between reasoning and argument, and I will not do so beyond this point.

Walton's concept of warrant is derived from Toulmin (1958/2003, p. 90–92), who analyzes an argument as consisting of (at least) a conclusion, a warrant, and data. Toulmin notes that the warrant is usually implicit; it is a general rule that allows the argument to move from the data to the conclusion. Toulmin also argues (p. 93) that an argument may require a qualifier for the conclusion, such

[4] I became aware of this phrase in personal conversations with or in public lectures by Ted Nelson when he was collaborating with a research group that I belonged to at the University of Jyväskylä sometime around the turn of the millennium. It is, I think, an autological phrase in that retroactive obviousness is itself a retroactively obvious idea. Considering that a Google search finds only two web pages predating 2007 containing that phrase, one written by Nelson (1999) and the other a collection of quotes from him, it was likely coined by him. A related phrase "retroactively plausible" is mentioned, however, by the OED in a 1950 quotation (*retroactive, adj.* 2014, at "retroactively, adv")

as "necessarily" or "probably", or exceptions to its validity to be spcified, as well as (p. 95–97) backing for the warrant.

For the purooses of this dissertation, I will be using a merged and simplified analysis: an argument consists of premises, an optional qualifier, and a conclusion. Premises include Walton's premises and warrants, as well as Toulmin's data, warrants and backing. Toulmin's exceptions I will include in the conclusion. The qualifier is often, but not always, implicit.

It is traditional to divide arguments (or sequences of reasoning) into two classes, *deductive* and *inductive*. Arguments that assert that the conclusion follows necessarily from the premises are deductive (in effect, they are arguments whose Toulmin qualifier is "necessarily" or similar); all other arguments are inductive. (the precise definition of these terms is disputed; see, e. g., Wilbanks 2010, and references therein). The theoretical study of valid forms of arguments abstracting from the content of the arguments comprises *(deductive) logic*, large parts of which is utterly standard, including but not limited to textbook classical logic (see, e. g., Burgess 2009); a similar theory of inductive arguments is a matter of controversy (see, e. g., Howson and Urbach 2006; Norton 2010; Worrall 2010). The theory of inductive reasoning is discussed more thoroughly in Chapter 4.

Philosophical reasoning has much in common with mathematical reasoning; deductive arguments are very common in the former and nearly universal in the latter. In fact, a precise boundary is hard to place. I personally find it useful to locate the boundary as follows: philosophy (paraphrasing Sellars 1962) deals with objects in the widest sense of that word—including physical objects like chairs and historical persons like Socrates as well as imaginary concepts like "the present King of France" (Russell 1905, p. 479) and abstract concepts like death—while mathematics deals with formal abstractions like numbers, sets and functions;[5] a mathematical argument functions entirely within the domain of mathematical objects, while a philosophical argument may show the identifiability of a nonmathematical concept with a certain mathematical one (I will discuss one example of this in Section 3.2). However, in truth, a precise demarcation does not really matter for this present work.

The study of the foundations of mathematics from the late 19th Century onward has created a sharper demarcation in that it is now standard to define the correctness of a mathematical argument with the potential ability to translate it into an inference in some formal system (see, e. g., Hilbert 1925/1967, p. 381–382; Bourbaki 1970/2004, p. 8; Mac Lane 1986, p. 377).[6] If one accepts formal inference as the ideal of both mathematical and philosophical argumentation, then a modern difference between philosophical and mathematical argument is

[5] Thurston (1994) defines mathematics as "the theory of formal patterns" or, alternatively, as the result of mathematicians recursively elaborating on "the natural numbers and plane and solid geometry" (p. 162).

[6] Common to both mathematical and philosophical argument is that, even when formalization is accepted as the ideal, the actual practice is a non-formalized argument, which Bundy et al. (2005) and MacKenzie (2005), writing about mathematical proof, call *rigorous*—an argument accepted as mathematical or philosophical by mathematicians or philosophers, respectively.

in the premises that are considered legitimate: in modern mathematics, premises must be previously declared axioms or definitions or else they must be previously proven theorems, while in philosophy, a much wider set of premises are admitted (and the taxonomy of all such premises would be a futile task). This model, of course, omits an explanation of where axioms and definitions come from; some of them come from philosophical analysis, some might be the result of idle speculation, but, like Lakatos (1976) demonstrates, many of them are proof-generated, the results of patching up faulty attempts at deductive proofs.[7]

One particular move that is common in philosophy and very rare in mathematics is *semantic ascent* (Quine 1960) or *going meta* (Dennett 2013), that is, moving from inside the problem to outside of it, the better to behold it in its entirety. This is why Passmore (1970) notes that a philosophical argument, although it often uses logic, is not typically itself a logical argument. A case in point is the *petitio principii*, which criticizes another argument for assuming that which it aims to prove; as Passmore points out, this is not to point out a logical fallacy (indeed, any argument for which the *petitio principii* applies is deductively valid) but to indicate the (lack of) relevance of the argument. Similarly, Ryle (1954) distinguishes mathematical proof from philosophical argument by noting that the latter typically operates at a metalevel compared to mathematics even when it concerns mathematical matters.

Another move common in philosophy is the *thought experiment*: a hypothetical (sometimes impossible) concrete situation explored in detail in order to expose consequences of a position under scrutiny. It is similar to the thought experiments that have been used in physics by, e. g., Albert Einstein (see, e. g., Norton 1991). This move is, in a restricted sense, common also in mathematics: the assumption of existence of a particular thing in order to show that this leads to a contradiction, leaving the nonexistence of that thing the only possible option. In philosophy, a thought experiment is usually not restricted to such deductive arguments; instead it is typically used as an *intuition pump* (Dennett 1980; Dennett 2013), a device for eliciting in the mind of the audience an intuitive belief in a specific proposition regarding the analysandum. At least Stitch (2001) argues that this method goes back to Plato.

A very famous instance of this move concerns the conventional analysis of knowledge as justified true belief (JTB). Edmund Gettier (1963) presented two thought experiments, the key point of which was to elicit the intuition that, in these specific cases, there is no knowledge even though JTB is established, from which Gettier infers that JTB is not a sufficient analysis of knowledge. In the literature, these and other similar thought experiments are typically called "Gettier cases" (Hazlett 2015).

The legitimacy of this move in philosophical analysis has recently been under debate (see, e. g., Brendel 2004; Williamson 2007; Nagel 2012; Mizrahi 2014).

---

[7] Andreas Stefik points out (in his comments attached to his review of this dissertation) another difference: he notes that mathematics is often used for prediction and doubts that philosophy is. This is, in my view, a true distinction between them. As I have pointed out above, when the mathematics begins, the philosophy (if any) has been concluded.

One might (like Mizrahi 2012, 2013) regard it as an appeal to (the writer's own) authority or question whether different people can be expected to have similar reactions to a particular intuition pump. As to the first point, in my view, an intuition pump is better understood as a defeasible argument (as in, if you do not share my intuition, then this argument will not convince you) than as an appeal to authority (if you do not share my intuition, then you are defective). The second point is valid as far as it goes, although there is empirical evidence to suggest that intuitions are, in fact, shared widely in at least some cases (e. g., Nagel et al. 2013; Seyedsayamdost 2015); but, more importantly, it again merely points towards the move being defeasible.

An interesting analogy between mathematical and philosophical argument arises from the Lakatos (1976) model of evolution of mathematical concepts and proofs. Lakatos suggests starting with a plausile conjecture and continuing with a thought experiment (which at least in the case of a geometrical conjecture can be very physical in nature), which results in a first attempt at a proof by decomposing the conjecture into a series of plausible lemmas. By an iterative process of discovering counter examples and improving both the conjecture and the proof in response, over and over again, one eventually arrives at a formal theory, that is, a set of axioms and definitions, in which a deductively valid proof is possible. Similarly, a philosophical argument should be viewed as a proof attempt, subject to revision (or the abandonment of a position) as counterexamples are discovered but, due to the nature of the subject matter, never maturing into a formal theory.[8]

Frogel (2005, p. 17–18) argues that a philosophical argument is essentially a kind of rhetoric, distinguished from other kinds of rhetoric by the presence of "metaphysical wonder" and an attempt at justification, as well as the absence of "binding criteria of justification". In a philosophical argument, Frogel posits, the speaker is "the man of truth [. . .] one whose fear of illusion is what motivates and drives him, as the sole thing guiding his thinking." (p. 57) However, the intended reader, the one "whose agreement is supposed to serve as a guarantee for truth" (p. 74), is, in Frogel's view, always the philosopher themself, and the greatest danger to a philosopher is self-deception. Yet, another philosopher reading their work cannot take their self-agreement as evidence of truth, but must independently reexamine the issue, and this, Frogel claims, explains why philosophers engage largely in refutations and rarely agree with each other.

Although it may seem quite outrageous to assert, like Frogel does, that a philosophical analysis is always written first for the writer themself, and only second for the actual reader, this is, in my view, very likely correct. Recall the anonymous philosopher who told Hyland (2001b, p. 217, quoted on p. 18) that using 'I' in philosophical writing lets the reader know what the writer is personally committed to. Conversely, a position the writer does not hold themself is in a weaker position to start with than a position they do hold.

---

[8] If one reduces a philosophical argument into a formal theory, the connection to the original subject matter (which is not formal) necessarily is severed. This is, perhaps, why the young Wittgenstein, after having formalized philosophy, concluded in the *Tractatus* (1921/1974) that it is all nonsense.

All of this results, unsurprisingly, a proliferation of positions on any given issue with no clear consensus resolution in sight. Johnstone (1963) goes so far as to argue that expecting complete agreement on a philosophical question is to misunderstand the question. There is, however, a sort of progress in philosophy, that of developing of distinctions and of sharpening of positions in response to critique, and a rough consensus of a position having been discredited sometimes occurs (see, e. g., Hacker 2009; Plant 2012; Olsen 2014). Similar progress can probably be expected in applied uses of philosophical analysis as well, if those who disagree with an analysis write a counterargument and publish it, resulting in a scholarly debate.

3.1.5 Summary

Analysis is a tool for clearing up conceptual confusion. An analysis asserts some sort of equivalence between a concept, the analysandum, and a bundle of concepts, the analysans. Classically, this equivalence is coextensionality, that is, the analysandum and analysans are asserted to have the same extension, while their intensions differ. Sometimes, instead of a coextensionality analysis, an explication is offered, in which the concept to be explicated, the explicandum, is asserted to be replaceable by a better concept, the explicatum, without asserting equivalence of any kind.

Analysis and explication usually require a supporting argument. The arguments presented in philosophical arguments are very similar to arguments used in mathematics, but there are differences. First, acceptable premises are not limited to axioms and previously proved theorems; often, premises are justified by the use of, e. g., thought experiments. Second, philosophical arguments go meta quite commonly. Third, philosophical argument is rhetoric intended to convince the author in the first instance, inviting both agreement and reasoned disagreement among the readers.

3.2 Analysis in computing

When Alan Turing (1937) introduced the model of computation that has since became known as a Turing machine, what he did was in essence to engage in philosophical analysis of the concept of computation or, more precisely, in its Carnapian explication. In this case, the analysandum or the explicandum is computation, and its analysas or explicatum is the Turing machine. Let us consider the four requirements Carnap (1962, p. 7, see also page 47 of this dissertation) posed. It is easy to see how the Turing machine is exact, fruitful (at least in retrospect) and simple enough. The question of similarity requires an argument, which Turing offered in his Section 9.

The main thrust of Turing's argument is that the Turing machine can be

obtained from a human computer[9] by applying a series of (informal) simplifying transformations, such as restricting the computer to a one-dimensional paper, and supposing that the set of a particular computer's possible relevant states of mind is finite. Even though these transformations remove the explicatum quite far from the actual behaviour of a computer, they are "not one[s] which seriously affect[] computation" (Turing 1937, p. 250), and thus do not compromise, in his view, the similarity of the explicatum to the explicandum.

Although the resulting explicatum is a mathematical model, Turing's argument is not mathematical in nature, as he himself points out: "All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically." (p. 249) This is not surprising, as a mathematical argument requires that both the analysandum and the analysans are precisely given at the outset, if not commonly understood as such then at least stipulated for the purposes of the argument. Quite to the contrary, a philosophical analysis like Turing's is fundamentally prior to any mathematical analysis: mathematics can only begin once the philosophy has been concluded.[10]

It is instructive to contrast Turing's analysis with that of Alonzo Church (1936), whose preferred explicatum of computability was the lambda calculus. Considering the Carnapian criteria, the explicatum was again (at least in retrospect) exact, fruitful and simple enough. On the question of similarity, Church offered two mathematical arguments in his Section 7: he first defined an algorithm and then showed that the lambda calculus can express every algorithm and nothing else; then he showed that under certain assumptions about the logic used, a theorem stating the value of a function given an argument can be proven if and only if that function is computable. He further, in footnote 3 on page 346, points out that the extensional equivalence of lambda calculus with the theory of general recursion had been proven (Turing had not yet published his paper), which suggested to him that they both describe the same natural concept, that of computability.

The striking difference between Turing and Church is that Church offered a quite weak philosophical analysis, basically just what Soare (2007) calls "evidence", while Turing offers a simple, direct and convincing philosophical analysis of the concept of computation. Even Church (1937, p. 43) regarded Turing's analysis as "making the identification with effectiveness in the ordinary (not explicitly defined) sense evident immediately"—in other words, convincing.

Philosophical analysis is, in a non-self-conscious way, a commonly applied technique in programming language research. For example, Adams (2015) recently proposed an analysis of the concept of macro hygiene (the capability of

9 At the time Turing was writing, the word "computer" was the name of a profession, not of a machine (cf. Grier 2007).

10 This claim of mine is in some apparent tension with the model of mathematics offered by Lakatos (1976), who rightly points out that the formalistic view of mathematics is only about a century old. However, the crucial distinction is, in my view, between arguments that are intended to be abbreviated statements of formal proofs and those that cannot be so construed; whether I am correct to label them as mathematics and philosophy, respectively, makes little difference to my larger argument.

a macro system to protect against unintentional variable capture) as it appears in Scheme, with the explicit goal of explicating a formal definition of the concept that can be used to compare implementations. His offered explication meets the exactness criterion without difficulty, and it presumably is simple enough. Whether it is fruitful remains to be seen; Adams does not himself offer any developed use cases for the explication. An argument that it is similar to the explicandum is a key part of the paper.

A number of researchers (e. g., Lieberman 1986; Snyder 1986; reviewed by Taivalsaari 1996) analyzed the concept of inheritance in the mid-1980s to early 1990s. Similarly, Harrison and Ossher (1993) introduced and analyzed the idea of "subject-oriented programming"; and Agerbo and Cornils (1998) analyzed the concept of design patterns. Some of these fulfill Carnap's criteria of explication, except that similarity to an existing concept does not usually come into play, as these analyses tried to explicate a new concept, or a deliberately different variant of an existing concept. In this, analysis in computing clearly differs from analysis in philosophy. Nevertheless, the elements of a Carnapian explication are, with caveats, present.

Another, quite different but noteworthy exemplar is the extensive review and synthesis of the conception of computing or computer science (however it is named) as a discipline by Tedre (2006); he is explicitly anarchist and multidisciplinary in his research approach, and a significant component of it is what I would call philosophical analysis. Similarly, Hanenberg (2010c) offered an analysis of (what he considers to be) proper research in software engineering and programming languages.

I should note that, while Rayside and Campbell (2000) discussed the relationship of object-oriented programming to Aristotle's philosophy, it is not the kind of philosophical analysis that I am discussing in this chapter. This sort of exegesis and reinterpretation of a single author is common in philosophy but, in my view, analysis is about what the analyst themselves hold to be true, not what they believe some other person has held to be true or its relation to another subject, in this case object-oriented programming.

3.3 Analysis in this dissertation

In this dissertation, analysis is used to clarify the unclear. Often the analysandum is unclear because it does not (yet) (quite) exist, though there is reason to believe that it ought to; thus, my analysis is often less about declaring necessary and sufficient conditions describing what the analysandum *is* than about explicating what the analysandum *should be*. Such clarification cannot be done empirically: since the thing does not yet exist, there is nothing to be done empirically.[11] It

[11] Cf. Hume (1740/2003, p. 334, emphasis in the original): "I have always remark'd, that the author proceeds for some time in the ordinary way of reasoning [...]; when of a sudden I am surpriz'd to find, that instead of the usual copulations of propositions, *is*, and *is not*, I

is also not a mathematical question: to prove a theorem, one must first set up axioms, but the axioms themselves already settle the question, as the theorem states nothing that is not already present in the axioms (such is the nature of deductive inference).[12] Analysis is, I hold, the only adequate approach for this.

You might reasonably ask, what are the standards of validity to be applied to philosophical analysis. I reject such standards as advocated by Hume (1748/2011, last paragraph, quoted in fn. 2 on p. 16) or Wittgenstein (1921/1974, Prop. 6.53, quoted on p. 45), which would render all substantive philosophical analysis invalid. At the same time, I do acknowledge that there is a grain of truth in them, in that I do not expect a philosophical analysis ever to receive unanimous agreement: when an analysis is backed by a deductive argument, a person who disagrees with its conclusion need only repudiate one of its premises, and I cannot envision any nontrivial argument (philosophical or not) to rely solely on premises that cannot be denied; and when the argument is inductive, there is no need even to repudiate any of the premises. Thus, any standard requiring agreement with the conclusion would be equally inappropriate.

Very recently, Dittrich (2015) has opined on the appropriate standard for evaluating philosophical argumentation in software engineering:

> "The quality criteria for philosophical argumentation are subject to philosophical subdisciplines and in part also depending on the philosophical school the argument is contributing to. In the context of Software engineering, I propose to apply (a) rigour of argumentation and (b) relevance of results".

I fully endorse her proposal. However, she leaves unspecified what the rigour of argumentation means. Paseau (2015, footnote omitted) defines it as follows:

> "There are perhaps two main ways in which an argument can be rigorous. One is by setting out its premisses expliticly; the other is by taking small argumentative steps."

Paseau's main point, however, is that rigour (thus defined) "is not in itself epistemologically valuable"; he recommends to make an argument as rigorous as necessary but no more. But what is that point of necessity? In my view, rigour is satisfied if the dissenting reader is given a clear enough argument that they can identify relevant points of disagreement and formulate a reasoned counterargument. That standard can facilitate proper scholarly debate on the issue under analysis.

My discussion of philosophical analysis has thus far only mentioned inductive argumentation, even though it is a key technique. The next chapter will explicate it in detail.

---

meet with no proposition that is not connected with an *ought*, and *ought not*. [...] For as this *ought*, or *ought not*, expresses some new relation or affirmation, [...it] seems altogether inconceivable, how this new relation can be a deduction from others, which are entirely different from it."

12 This is not to say that mathematics and theorem-proving could not be used to illuminate the issue under analysis. I do this myself in this dissertation.

4 INDUCTIVE REASONING

In this dissertation, induction plays a double role. First, many of the arguments I advance are inductive in nature. Second, I hold empirical reasoning, which is of paramount importance to evidence-based practice, to be inductive in nature. In this chapter I will be presenting and arguing for a framework of inductive logic (which is, I hasten to add, not original to me). Regarding the first role, I offer this framework as my understanding of what inductive reasoning is and as my preferred standard for evaluating the inductive arguments I make in this dissertation, though I do not myself nor do I expect anyone else to actually go through the computations in most situations. Regarding the second role, I offer this framework as a fundament of evidence-based practice, particularly evidence-based programming language design as I introduce it in Chapter 10.

Deduction preserves truth: if the premises of a deductive argument are true, then its conclusion is as well. Or, in other words, the conclusion is true in every possible world where the premises are true. The focus of this chapter is reasoning where this does not necessarily hold, that is, *inductive* arguments. It seems clear that such arguments are not all equally strong: for example, "if the sun rises tomorrow, then the sun will rise the day after tomorrow" and "if the sun rises tomorrow, then the sun does not rise tomorrow" are both inductive arguments but the former is clearly stronger than the latter (in fact, the latter seems to be one of the weakest arguments one could make).

In this chapter, I will be arguing for a particular logic of induction. Specifically, I will be arguing that it is reasonable to regard a logic of induction as an interpretation of the theory of probability. Further, I will be arguing that any such argument evaluation strategy is recipient-dependent but that there are strategies to overcome the inherent subjectivity of this approach so that the strategy is applicable to evaluating scientific arguments.

This claim and the line of argument are not original to me. Philosophers and scientists who have argued for a close correspondence of a logic of induction and the theory of probability include (but are not limited to) Carnap (1962), Howson and Urbach (2006), Jaynes (2003), Jeffreys (1931), Keynes (1921/2014), and von Wright (1957). In particular, Howson and Urbach (2006) had a big influence in

my thinking in these issues, even though my approach here is somewhat different from theirs. Also, Hahn and Oaksford (2006) and Zenker (2013a) suggest using the theory of probability to evaluate argument strength in a somewhat similar fashion.[1]

4.1 An analysis short of probability

The argument I advance below aims to justify certain postulates about the extra-deductive evaluation of arguments that are quite similar to familiar axioms of probability, which then allows me to draw a correspondence between this evaluation system and the system of probability theory; this argument is influenced by, and is in many ways similar to, the justification of an axiom system for probability advanced by Cox (1946) and restated by Jaynes (2003). The most significant difference is that Cox and Jaynes start from the assumption that probabilities must be real numbers, while my argument defers that choice until fairly late in the process (for a survey of non-real-valued conceptions of probability, see Fine 1973). Further, Cox and Jaynes do not extend their argument to σ-algebras and countable additivity like I do below. Finally, framing the problem as argument strength analysis instead of the behavior of agents, like I do here, is uncommon.

For concreteness, I will discuss here a quite standard *quantifier-free first-order predicate language* $\mathcal{L}$.[2] This logic is two-valued and standard except for the omission of quantifiers (cf., e. g., Fitting 1996; Smith 2003; Ben-Ari 2012), and I will omit its formal definition. I will discuss quantifiers later in this chapter.

A *vocabulary* defines at least one predicate symbol (each with a positive finite arity), at least one constant symbol, and possibly any number of function symbols (each with a positive finite arity); the language $\mathcal{L}$ is implicitly parametrized by the vocabulary. The terms of the language comprise variables x, constant terms C, and function applications $F(t_1, \ldots, t_n)$; formulas comprise atomic formulas $R(t_1, \ldots, t_n)$,[3] negations $\neg p$, conjunctions $p \wedge q$, and disjunctions $p \vee q$. The material implication $p \to q$ may be used as another way of writing $\neg p \vee q$.

An *interpretation* $\mathcal{M}$ of the language defines a collection of individuals (sometimes called the *domain of discourse* or simply the *domain*) and for each constant symbol, function symbol, and predicate symbol an interpretation as an individual, as a function over individuals, and as a relation between individuals, respectively. Given an interpretation $\mathcal{M}$ and a variable assignment φ, which maps each variable to an individual of the interpretation, one can determine what individual $[\![t]\!]^{\varphi}_{\mathcal{M}}$ a term t stands for and whether a formula p is true (written $\mathcal{M} \vDash_{\varphi} p$) or false (written $\mathcal{M} \nvDash_{\varphi} p$). In case of terms and formulas containing no free vari-

1 Pfeifer (2013), however, defines argument strength in a similar probabilistic framework to be a rather different function.

2 Abusing the now-standard terminology of Karp (1965), it is a $L_{\omega 0}$ language.

3 In case the argument terms are constant symbols or variables, I will sometimes omit the parentheses and commas from both function applications and atomic formulas.

ables, the variable assignment is not needed and I will write $[\![t]\!]_{\mathcal{M}}$, $\mathcal{M} \models p$, and $\mathcal{M} \not\models p$. I will be calling formulas with no free variables *sentences* and the set of all sentences $\mathcal{S}$.

A brief digression on alternative terminology is in order. It is sometimes convenient to assume that the *real world* is one interpretation of the logical language, and the rest of the interpretations comprise exactly (up to isomorphism) the *counterfactual worlds*, which together with the real world then are the *possible worlds*; thus, one sometimes says that a sentence is true in the real world or that a sentence is true in all possible worlds. Now, this usage may suggest a naïve realist ontology where one assumes a unique, objective reality which, to the extent it is perceived at all, is perceived perfectly (see, e. g., Blackmore 1979), but there is no reason it must be read that way. Following Quine (1968) and Lewis (1979), one often views the world as being centered to a person (or other agent) at a particular point in time, and thus this framework easily allows multiple personal realities if it is desired. Similarly, a socially constructed reality can be easily accommodated, as well as a more sophisticated variant of realism. Interpretations are also often called *models*, especially if one wishes to emphasize that no interpretation is assumed to be (in a strong, ontological sense) the real world.

In many applications, there will be symbols in the signature whose meaning in all interpretations must be the same. For example, unless one is specifically studying the theory of real arithmetic, it is often expedient to restrict the vocabulary and interpretations under study so that every real number is both an individual and a constant symbol standing for itself, with the usual operations and relations of real arithmetic similarly available as function symbols and relation symbols standing for themselves, respectively. Following Kripke (1972), I will say that any symbol that stands for the same thing in every interpretation is *rigidly designated*.

An *argument* using this language consists of a finite or countably infinite set of sentences K and a sentence q; the sentences of K are the argument's *premises* and the sentence q is its *conclusion*; in symbols, I write an argument as $K \vdash q$ (where I usually omit the curly brackets in the premise set and use commas for both set union and set enumeration, without cause for confusion). An argument $K \vdash q$ is *valid* if there is no interpretation where the sentences of K are true and q is false; and it is *self-contradictory* if if there is no interpretation where the sentences of K are true and q is true. I will call an argument *inadmissible* if it is both valid and self-contradictory and *defeasible* if it is neither valid nor self-contradictory. Note that an argument that is not inadmissible—let me call such arguments *admissible*—has at least one interpretation that makes its premises true. There are well known deductive logics for languages of this kind that allow one to recognize an argument as valid or self-contradictory (see, e. g., Gödel 1930/1967; Henkin 1949); as my topic here is inductive logic, I will simply assume such a deductive logic without elaboration.

Notice that an argument is defined to consist of sentences. That means the underlying language does not have to be a predicate language; a propositional one would be sufficient in the abstract. Of course, a more powerful language

could also be used (and I will, later on). The reason I am using a predicate language is to emphasize that elementary sentences have an underlying structure and so that my later extension of the language can be carried out with minimal effort.

There are two obvious ways to frame the problem of evaluating the strength of an argument. One is to postulate an ordering relation between arguments, constrained by the logical character of the arguments. Another is to postulate a function from arguments to some set of strength values which possesses an ordering relation. As we can easily identify two such strength values (one assignable to valid arguments and another assignable to self-contradictory ones, which naturally ought to be the top and bottom elements with respect to the ordering—though one must be careful here with respect to non-admissible arguments), and as any such function will automatically define an ordering of actual arguments, the functional approach appears superior.

Thus, I will postulate a function $[\![]\!] : \mathcal{A} \to V$, where $\mathcal{A}$ is the set of all admissible[4] arguments and $(V, \leq)$ is a so far unspecified partially ordered set of strength values; let $V_* \subseteq V$ comprise those strength values actually produced by the valuation function. Now, V_* contains at least two elements, since valid admissible arguments should be assigned the same value $\top \in V_*$ (as they are of equal strength), that all self-contradictory admissible arguments should be assigned some other value $\bot \in V_*$, and that these values bound the strength valuation function in that the inequality $\bot \leq v \leq \top$ holds for all $v \in V_*$.

Under the preceding postulates it will be possible to use equivalence classes of arguments as the set of V_* and have $[\![]\!]$ translate each argument to its equivalence class. Such a trivial implementation would be cheating, however, as the idea is to have $[\![]\!]$ do the argument evaluation instead of pushing the evaluation problem down to the next level. However, the possibility of such a trivial implementation is not a problem; what is needed is to ensure that a nontrivial implementation remains possible. Thus, I will require of my explication that it may not depend on being able to recover the argument from its strength value.

Consider now the strength of the admissible argument $K \vdash p \wedge q$, where K is some (perhaps empty, perhaps infinite) set of sentences and where p and q are sentences. Note that $[\![K \vdash p \wedge q]\!]$ is not simply a function of $[\![K \vdash p]\!]$ and $[\![K \vdash q]\!]$, as $K \vdash p$ may in some cases come partway (or even completely) toward establishing q. For example, suppose q follows deductively from p; then, an argument establishing p also establishes $p \wedge q$. Suppose now, instead, that p and q are mutually incompatible; then, an argument establishing p will show $p \wedge q$ to be self-contradictory. These interactions between p and q cannot be derived from $[\![K \vdash p]\!]$ and $[\![K \vdash q]\!]$ unless it were allowed to recover the argument yielding the strength value, which I just forbade in the previous paragraph. But these interactions *are* embodied in the argument $K, p \vdash q$: if p deductively entails q, then

[4] I am restricting the valuation function to admissible arguments in order to exclude certain inconsistencies that flow from non-admissible arguments being simultaneously valid and self-contradictory. It is, however, not the only possible way to exclude them (for discussion, see Leblanc and Roeper 1989, n. 5 on p. 509).

this argument is valid, and if p and q are mutually incompatible, the argument is self-contradictory. Thus, it is clear that if $[\![K \vdash p \wedge q]\!]$ depends on $[\![K \vdash p]\!]$, it must also depend on $[\![K, p \vdash q]\!]$; symmetrically, if it depends on $[\![K \vdash q]\!]$, it must also depend on $[\![K, q \vdash p]\!]$. But $K \vdash p$ and $K, p \vdash q$ on the one hand and $K \vdash q$ and $K, q \vdash p$ on the other both are equally good paths to reach to $K \vdash p \wedge q$, and they thus ought to result in the same strength evaluation. Therefore, I will postulate a function $C : V \times V \to V$ satisfying the equation

$$C([\![K \vdash p]\!], [\![K, p \vdash q]\!]) = C([\![K \vdash q]\!], [\![K, q \vdash p]\!]) = [\![K \vdash p \wedge q]\!]. \tag{1}$$

The following properties are easily derived:

$$v \leq w \Rightarrow C(v, u) \leq C(w, u) \tag{2}$$
$$v \leq w \Rightarrow C(u, v) \leq C(u, w) \tag{3}$$
$$(v \leq w \vee w \leq v) \wedge C(v, u) = C(w, u) \Rightarrow v = w \tag{4}$$
$$(v \leq w \vee w \leq v) \wedge C(u, v) = C(u, w) \Rightarrow v = w \tag{5}$$
$$C(v, w) = C(w, v) \tag{6}$$
$$C(\bot, v) = C(v, \bot) = \bot \tag{7}$$
$$C(\top, v) = C(v, \top) = v \tag{8}$$
$$C(v, C(w, u)) = C(C(v, w), u) \tag{9}$$

In other words, $V(C)$ is a commutative monoid with $\top$ as the identity element.

Strictly speaking, these results are applicable only to those $v, w, u \in V_*$ for which an admissible argument $A_v, A_w, A_u \in \mathcal{A}$, for which $[\![A_v]\!] = v$, $[\![A_w]\!] = w$, and $[\![A_u]\!] = u$ hold, can be found that are suitable for use in Equation 1. There is no reason to think that they cover the full range V_*, not to mention the rest of V, but, as a pragmatic matter, I will generalize such results as postulates to the full domain where the result is defined.

Consider now the admissible argument $K \vdash p \vee q$ in the case where p and q are incompatible given K. In such a case, clearly the strength of the argument is a strictly increasing function of the strengths of $K \vdash p$ and $K \vdash q$. I will use D for that function; thus the following equation must hold:[5]

$$K \nvdash p \wedge q \Rightarrow D([\![K \vdash p]\!], [\![K \vdash q]\!]) = [\![K \vdash p \vee q]\!] \tag{10}$$

The following are a restatement of the strict increase requirement:

$$v \leq w \Rightarrow D(v, u) \leq D(w, u) \tag{11}$$
$$v \leq w \Rightarrow D(u, v) \leq D(u, w) \tag{12}$$

[5] I use here $K \nvdash p \wedge q$ as a shorthand for the statement that $K \vdash p \wedge q$ is self-contradictory.

The following additional properties are simple to establish (with the same caveat as with C):

$$(v \leq w \vee w \leq v) \wedge D(v,u) = D(w,u) \Rightarrow v = w \tag{13}$$
$$(v \leq w \vee w \leq v) \wedge D(u,v) = D(u,w) \Rightarrow v = w \tag{14}$$
$$D(v,w) = D(w,v) \tag{15}$$
$$D(\bot,v) = D(v,\bot) = v \tag{16}$$
$$D(v,D(w,u)) = D(D(v,w),u) \tag{17}$$

These establish $V(D)$ as a commutative monoid with $\bot$ as the identity element.

Consider now the admissible argument $K \vdash a \wedge (b \vee c)$, where b and c are incompatible given K. It can be evaluated directly:

$$\begin{aligned}[\![K \vdash a \wedge (b \vee c)]\!] &= C([\![K \vdash a]\!], [\![K, a \vdash b \vee c]\!]) \\ &= C([\![K \vdash a]\!], D([\![K, a \vdash b]\!], [\![K, a \vdash c]\!])).\end{aligned}$$

Or it can be evaluated indirectly (note that by hypothesis, $a \wedge b$ and $a \wedge c$ are incompatible given K):

$$\begin{aligned}[\![K \vdash a \wedge (b \vee c)]\!] &= [\![K \vdash (a \wedge b) \vee (a \wedge c)]\!] \\ &= D([\![K \vdash a \wedge b]\!], [\![K \vdash a \wedge c]\!]) \\ &= D(C([\![K \vdash a]\!], [\![K, a \vdash b]\!]), C([\![K \vdash a]\!], [\![K, a \vdash \wedge c]\!])).\end{aligned}$$

This suggests the following general distributivity rule:

$$C(v, D(w,u)) = D(C(v,w), C(v,u)) \tag{18}$$

which, together with previous rules, establishes $V(D,C)$ as a commutative semiring. Note that there is no similar distributivity rule for $D(v, C(w,u))$; the asymmetry (which is not present in $\mathcal{L}$) arises from the incompatibility requirement of D.

For pragmatic reasons it makes sense to further require $V(D,C)$ to be a field by requiring the existence of inverses for both D and C; this allows equations involving C and D to be solved. With that addition, I will summarise the current analysis:

Analysis 2. *The strength of an admissible argument is evaluated by a function* $[\![]\!] : \mathcal{A} \to V$*, where* $V(\leq)$ *is a partially ordered set containing at least two elements* $\top$ *and* $\bot$ *and* $V(D,C)$ *is a field; all of which obey the following rules:*

$$C \text{ and } D \text{ are strictly increasing}$$
$$K \vdash p \text{ is valid} \Rightarrow [\![K \vdash p]\!] = \top$$
$$K \nvdash p \Rightarrow [\![K \vdash p]\!] = \bot$$
$$\bot \leq [\![A]\!] \leq \top$$
$$C([\![K \vdash p]\!], [\![K, p \vdash q]\!]) = C([\![K \vdash q]\!], [\![K, q \vdash p]\!]) = [\![K \vdash p \wedge q]\!]$$
$$K \nvdash p \wedge q \Rightarrow D([\![K \vdash p]\!], [\![K \vdash q]\!]) = [\![K \vdash p \vee q]\!]$$

If one further requires the ordering relation to be total (although it is not at all clear that one must do so), the resulting structure is an ordered field, which means that it includes an isomorphic image of the rational numbers; at that point, there isn't much to be gained from not adding the irrational numbers as well and identify $V = \mathbb{R}$. The next question is the choice of C and D.

Cox (1946) and Jaynes (2003) argue for the identification of C with ordinary multiplication by postulating that C has continuous second derivatives and then deducing from the associativity property (9) that C satisfies the equation $cf(C(a,b)) = f(a)f(b)$; arbitrarily choosing $c = 1$ and $f = id$, they arrive at the identification of C with multiplication.[6] This argument seems to me underwhelming: that multiplication has the associativity property is manifest in the standard axiomatization of $\mathbb{R}$, and thus there is no need to resort to concepts like continuity to prove that fact; further, the argument does not allow one to exclude other seemingly obvious choices like addition, which is another operation possessing the associativity property and which is allowed in the equation by $f(x) = e^x$.

A better argument starts from noticing that D and C have to be chosen together, because of their deep interconnections manifested in their ordered field nature. There are, of course, a large number of ways to choose D and C (which then compel choices for $\top$ and $\bot$), but the simplest is to assign $D = +$ and $C = \times$ (and consequently $\bot = 0$ and $\top = 1$); there being no apparent reason not to take the simplest choice, it is the one I will choose. Now, the analysis can be restated:

Analysis 3. *The strength of an admissible argument is evaluated by a function* $[\![]\!] : \mathcal{A} \to [0,1]$ *obeying the following rules:*

$$K \vdash p \text{ is valid} \Rightarrow [\![K \vdash p]\!] = 1$$

$$K \nvdash p \Rightarrow [\![K \vdash p]\!] = 0$$

$$[\![K \vdash p \wedge q]\!] = [\![K \vdash p]\!]\,[\![K, p \vdash q]\!] = [\![K \vdash q]\!]\,[\![K, q \vdash p]\!] \tag{19}$$

$$K \nvdash p \wedge q \Rightarrow [\![K \vdash p \vee q]\!] = [\![K \vdash p]\!] + [\![K \vdash q]\!] \tag{20}$$

A simple rule for negation can be derived. Since p and $\neg p$ are incompatible, Equation (20) can be used to produce $1 = [\![K \vdash p \vee \neg p]\!] = [\![K \vdash p]\!] + [\![K \vdash \neg p]\!]$; and thus

$$[\![K \vdash p]\!] = 1 - [\![K \vdash \neg p]\!] \tag{21}$$

holds. Now, define that p and q are *equivalent given* K if for every interpretation that makes the sentences in K true, p and q are both true or both false in that interpretation. The following is then a simple corollary of Equation 21:

$$p \text{ and } q \text{ are equivalent given } K \Rightarrow [\![K \vdash p]\!] = [\![K \vdash q]\!]\,. \tag{22}$$

This equation can also be justified by noting that equivalent sentences have the same content and therefore they ought to be interchangeable with respect to argument strength. Such an argument justifies also the following:

$$K \text{ and } K' \text{ are equivalent} \Rightarrow [\![K \vdash p]\!] = [\![K' \vdash p]\!]\,. \tag{23}$$

[6] Like Jaynes (2003) notes, an equivalent result can be proven with just an assumption of strict increase and continuity (see, e. g., Aczél 2004).

4.2 Connection to probability theory

There is a simple translation of this machinery into the mathematical theory of probability (cf. Gaifman 1964; Scott and Krauss 1966; Gaifman and Snir 1982). Let $\mathbb{M}$ be the class of all interpretations of $\mathcal{L}$ whose collection of individuals is a set (that is, those interpretations that are not in a set-theoretic sense too big). One can further define a function $\mathcal{T}\colon \mathcal{S} \to \mathcal{P}(\mathbb{M})$ by the following equation:

$$\mathcal{T}(p) = \{\, \mathcal{M} \in \mathbb{M} \mid \mathcal{M} \vDash p \,\}. \tag{24}$$

The following propositions follow from the definitions:

$$\mathcal{T}(\neg p) = \mathbb{M} \setminus \mathcal{T}(p) \tag{25}$$
$$\mathcal{T}(p \vee q) = \mathcal{T}(p) \cup \mathcal{T}(q) \tag{26}$$
$$\mathcal{T}(p \wedge q) = \mathcal{T}(p) \cap \mathcal{T}(q) \tag{27}$$

From these follow trivially that the range of $\mathcal{T}$, that is

$$\operatorname{ran}\mathcal{T} = \{\, \mathcal{T}(s) \mid s \in \mathcal{S} \,\}, \tag{28}$$

contains $\mathcal{P}(\mathbb{M})$ as a member and is closed under complementation, finite unions, and finite intersections; it is, therefore, an algebra on $\mathbb{M}$.

I will call $\mathcal{L}$ a *finite language* if it has no function symbols and a finite number of both constant and predicate symbols. Such a language has a finite number of atomic sentences.

Proposition 4. *If $\mathcal{L}$ is a finite language, then* $\operatorname{ran}\mathcal{T}$ *is finite.*

Proof. Define for each sentence p a function $f_p\colon \mathbb{M} \to \{0,1\}$ so that

$$f_p(\mathcal{M}) = \begin{cases} 1, & \text{if } \mathcal{M} \vDash p \\ 0, & \text{if } \mathcal{M} \nvDash p \end{cases}$$

Let $q_1, \ldots, q_n$ be the atomic sentences of a finite $\mathcal{L}$. It is a simple matter to construct for each sentence p a function $g_p : \{0,1\}^n \to \{0,1\}$ such that $f_p(\mathcal{M}) = g_p(f_{q_1}(\mathcal{M}), \ldots, f_{q_n}(\mathcal{M}))$. Notice now that $\mathcal{T}(p) = \{\, \mathcal{M} \in \mathbb{M} \mid f_p(\mathcal{M}) = 1 \,\}$. But the set $\{\, g_p \mid p \in \mathcal{S} \,\}$ is finite; thus the set $\{\, f_p \mid p \in \mathcal{S} \,\}$ must also be finite, as must $\operatorname{ran}\mathcal{T}$. □

The corollary to this proposition is that in a finite language $\operatorname{ran}\mathcal{T}$ is trivially a σ-algebra.

Now, for all sets of propositions K define $\mathcal{T}_K\colon \mathcal{S} \to \mathcal{P}(\mathbb{M})$ as

$$\mathcal{T}_K(p) = \mathcal{T}(p) \cap \left(\bigcap_{q \in K} \mathcal{T}(q) \right). \tag{29}$$

Since the intersection of (σ-)algebras is a (σ-)algebra, $\operatorname{ran}\mathcal{T}_K$ is an algebra and, when $\operatorname{ran}\mathcal{T}$ is a σ-algebra, $\operatorname{ran}\mathcal{T}_K$ is as well. Now, define $P_K\colon \operatorname{ran}\mathcal{T}_K \to \mathbb{R}$ as

$$P_K\left(\mathcal{T}_K(p)\right) = [\![K \vdash p]\!] \tag{30}$$

for all sentences p. For P_K to be well defined, $\mathcal{T}_K(p) = \mathcal{T}_K(p') \Rightarrow [\![K \vdash p]\!] = [\![K \vdash p']\!]$ must hold for all sentences p and p'; but this follows directly from Equation (22). It is not difficult to see that P_K is nonnegative and finitely additive (and in the case of a finite language, also trivially countably additive). Since $\mathcal{T}_K(p \wedge \neg p) = \emptyset$ and $[\![K \vdash p \wedge \neg p]\!] = 0$, it follows that $P_K(\emptyset) = 0$ holds. A similar argument shows that $P_K(\operatorname{ran}\mathcal{T}_K) = 1$. Therefore, P_K is a finitely additive probability measure (and, in the case of a finite language, a probability measure).

Recall that the corresponding conditional probability is standardly defined as

$$P_K(A \mid B) = \frac{P_K(A \cap B)}{P_K(B)} \tag{31}$$

for all $A, B \in \operatorname{ran}\mathcal{T}_K$ such that $P_K(B) \neq 0$. The following proposition shows that this standard conditional probability corresponds to having the condition as a premise in an argument:

Proposition 5. *For all $p, q \in \mathcal{S}$ such that $P_K(\mathcal{T}_K(p)) \neq 0$, the equation*

$$P_K(\mathcal{T}_K(q) \mid \mathcal{T}_K(p)) = [\![K, p \vdash q]\!] \tag{32}$$

holds.

Proof.

$$\begin{aligned} P_K(\mathcal{T}_K(q) \mid \mathcal{T}_K(p)) &= \frac{P_K(\mathcal{T}_K(q) \cap \mathcal{T}_K(p))}{P_K(T_K(p))} = \frac{[\![K \vdash q \wedge p]\!]}{[\![K \vdash p]\!]} = \frac{[\![K \vdash p]\!]\,[\![K, p \vdash q]\!]}{[\![K \vdash p]\!]} \\ &= [\![K, p \vdash q]\!]\,. \end{aligned}$$

□

To make $\operatorname{ran}\mathcal{T}_K$ a σ-algebra even in the case of an infinite language requires the extension of the language to handle countably infinite disjunctions and conjunctions (cf. Scott and Krauss 1966). Define a new language $\mathcal{L}_\omega$ as follows: its terms are the terms of $\mathcal{L}$ and its formulas are the formulas of $\mathcal{L}$ with two new kinds of formulas $\bigvee Q$ and $\bigwedge Q$, where Q is a finite or countably infinite set of formulas of $\mathcal{L}_\omega$. Common variations of these two notations are used freely so long as no confusion arises. The interpretations of $\mathcal{L}_\omega$ are the interpretations of $\mathcal{L}$; truth and falsity is defined as in $\mathcal{L}_\omega$ with the addition that the formula $\bigvee X$ or $\bigwedge X$ is true in an interpretation $\mathcal{M}$ and a variable assignment φ if and only if $M \vDash_\varphi p$ holds for some or all $p \in X$, respectively. This language is an $L_{\omega_1 0}$ infinitary language in the now-standard terminology of Karp (1965). All the definitions and results stated above for $\mathcal{L}$, except for the definition of a finite language, carry over to $\mathcal{L}_\omega$ unchanged; $\mathcal{L}_\omega$ is never a finite language.

With this extension of the language, two more propositions follow from the definitions (where K and Q are finite or countably infinite sets of sentences):

$$\mathcal{T}_K\left(\bigvee Q\right) = \bigcup_{q\in Q} \mathcal{T}_K(q) \tag{33}$$

$$\mathcal{T}_K\left(\bigwedge Q\right) = \bigcap_{q\in Q} \mathcal{T}_K(q) \tag{34}$$

Thus, $\operatorname{ran}\mathcal{T}_K$ is closed under countably infinite unions and intersections, which makes it a σ-algebra. If one generalizes Equation (20) as follows—

$$\text{Sentences in } X \text{ are pairwise incompatible} \Rightarrow \left[\!\!\left[K \vdash \bigvee X\right]\!\!\right] = \sum_{p\in X} [\![K \vdash p]\!] \tag{35}$$

—then P_K is countably additive and therefore a probability measure.

At this point, I have developed enough machinery to argue that the strength of an inductive argument (based on finitary or infinitary quantifier-free predicate language) follows the axioms of the standard theory of probability calculus.

4.3 Interpretations

The above analysis does not specify a unique argument strength valuation function. To see why, consider a signature with one predicate symbol R, which is unary, one constant symbol c and no function symbols. A simple mathematical induction demonstrates that every sentence of $\mathcal{L}$ with that signature is deductively equivalent to Rc, $\neg Rc$, $Rc \wedge \neg Rc$, or $Rc \vee \neg Rc$. It is similarly easy to see that the all sets of sentences are deductively equivalent to $\emptyset$, $\{Rc\}$, $\{\neg Rc\}$, or $\{Rc, \neg Rc\}$, and only the first three can be used as premises in an admissible argument. The same holds for sentences of $\mathcal{L}_\omega$ with that same signature, as well. Below, I enumerate all admissible arguments, up to deductive equivalence, giving the constraints on their strength valuation that are derivable from the analysis of argument strength valuation:

$$\begin{aligned}
[\![\emptyset \vdash Rc]\!] &= a & [\![\emptyset \vdash \neg Rc]\!] &= 1 - a \\
[\![Rc \vdash Rc]\!] &= 1 & [\![Rc \vdash \neg Rc]\!] &= 0 \\
[\![\neg Rc \vdash Rc]\!] &= 0 & [\![\neg Rc \vdash \neg Rc]\!] &= 1 \\
[\![\emptyset \vdash Rc \wedge \neg Rc]\!] &= 0 & [\![\emptyset \vdash Rc \vee \neg Rc]\!] &= 1 \\
[\![Rc \vdash Rc \wedge \neg Rc]\!] &= 0 & [\![Rc \vdash Rc \vee \neg Rc]\!] &= 1 \\
[\![\neg Rc \vdash Rc \wedge \neg Rc]\!] &= 0 & [\![\neg Rc \vdash Rc \vee \neg Rc]\!] &= 1
\end{aligned}$$

As can be seen, there is one free parameter $a \in [0, 1]$, any value of which produces a different valuation function for argument strength. This argument becomes more complicated for more complex signatures but adding more symbols does not add new constraints on $[\![\emptyset \vdash Rc]\!]$ and thus does not cause it to be uniquely determined.

Consider the concrete case where the individuals are taken to be future coin tosses and R is the property of a coin toss landing on head. If I believe the coin to be fair and fairly tossed, I ought not to favor either outcome, and thus hold $a = 1 - a$, which yields $a = \frac{1}{2}$. However, if the individuals are rolls of a single die and R is the property of such a dice roll resulting in 1, then by a similar line of reasoning, $a = \frac{1}{6}$. One can, of course, choose other values of a if one does not believe the situation to be fair. Similarly, this reasoning will not apply if the individuals are something other than coin tosses or rolls of dice. Thus, there can be no unique a that depends solely on the form of the argument; the semantic content of an argument is relevant to its strength.

The strength of an argument is ultimately a measure of its ability to convince another person. The axioms of probability leave, in the example above, a parameter a to be chosen freely, and this freedom gives room for individual variation. Given the semantic content of $\emptyset \vdash \neg Rc$ as stating that in a particular toss the coin does not land on head, one individual might value this argument at $\frac{1}{2}$ on fairness grounds, another at or close to 0 because they think the toss is fraudulent, and yet another might value it at $\frac{3}{4}$ because that number gives them a warm, fuzzy feeling. This leads to the following analysis:

Analysis 6. *The strength of an argument is a subjective evaluation, by the person to whom it is offered, of its ability to convince them, subject to the constraints of probability theory.*

The constraint offered by probability theory is not inconsequential. For example, once one has freely and subjectively chosen $[\![\emptyset \vdash \neg Rc]\!] = \frac{3}{4}$, one is compelled to adopt $[\![\emptyset \vdash Rc]\!] = \frac{1}{4}$; anything else would be incoherent or irrational. This is very similar to the situation where probability is interpreted as a person's *credence* or *degree of belief* on a proposition, a position commonly known as *personalism*, *probabilism*, or *Bayesianism* (see, e. g., Jeffrey 2004; Easwaran 2011a,b). In that approach, a person is viewed to possess a real-valued *credence function* $P(\cdot \mid \cdot)$, which is defined on all pairs of conceivable propositions where the second proposition is not self-contradictory. Further, upon receiving new information d that they take to be certain, a person is supposed to update the credence function by setting (simultaneously for all sentences p) $P(p) := P(p \mid d)$. Other updating rules have been proposed to deal with uncertain data.

Various arguments are standardly advanced to show that it is both necessary and sufficient for rationality that the credence function is a conditional probability:[7] the argument advanced by Cox (1946), which I paralleled earlier in this chapter, is one; another is based on a theorem stating that if a credence function is not a conditional probability, then there is another credence function that is a conditional probability which is also closer to the truth (Joyce 1998); and yet another, the *Dutch book argument* (see e.g. Ramsey 1926/1999; de Finetti 1937/1992; Lehman 1955; Kemeny 1955; Shimony 1955; Jeffrey 2004; Easwaran 2011a), relies

[7] Here, I do not mean to say that the credence function is defined by Equation (31) but that it satisfies axioms of conditional probability, similar to my Analysis 3, from which the Kolmogorov axioms with respect to the first argument and Equation (31) follow as theorems.

on a theorem showing that the credence function being a conditional probability is necessary and sufficient for a person betting according to it to be invulnerable to a set of bets that will certainly net a loss. The technical details of these arguments are beside my point here and are therefore omitted.

Bayesianism suffers from three well-known problems which are absent from the approach I have adopted in this chapter. Since the axioms of probability assign probability 1 to all valid sentences, a Bayesian agent is *logically omniscient*, that is, it knows and takes into account all logical truths, which makes a Bayesian agent unlike any concrete agent, real or conceivable (see, e. g., Smithies 2015). Relatedly, a Bayesian agent cannot realize belatedly that a previously learned piece of evidence is relevant to a particular hypothesis (this is known as the *problem of old evidence*; see, e. g., Wenmackers and Romeijn 2015). Finally, a Bayesian agent cannot invent a new theory with new theoretical terms (see, e. g., Wenmackers and Romeijn 2015). These problems are all caused by the choice to model the temporal behavior of an agent, and they disappear when merely an atemporal logic is considered (cf. Howson and Urbach 2006, Ch. 3).

Earlier, I suggested that one might regard a coin toss fair and thus assign $[\![\varnothing \vdash \neg Rc]\!] = \frac{1}{2}$, while another might be dogmatically convinced that the coin toss is fraudulent, assigning $[\![\varnothing \vdash \neg Rc]\!] = 0$. I treated these these characterizations of the tossing process as meta-premises constraining the argument strength; however, characterization of a coin toss as fair is not really a statement about the argument but about the physical process. Popper (1959) argued that certain physical processes can be described as possessing a physical constant called *propensity* or *chance* which constrains the random behavior of the process and which satistifies the axioms of probability. A coin toss seems to qualify as such a random process, and saying that it is fair is just a statement that the coin toss's propensity to produce heads, or the chance of heads in the coin toss, is $\frac{1}{2}$.

Following Lewis (1980) and Meacham (2010), if the conclusion of an argument states the particular outcome of a physical chance process and its premises concern only with either the past of that outcome or the general theory of the chance process and if the premises imply a particular chance for that outcome, then the argument strength should be that chance; Lewis called this the *principal principle*. Suppose I add to the language rigidly designated constants for real numbers between zero and one, inclusive, and the new unary predicate symbol C, to be understood as stating that the argument—if it denotes a real number between zero and one, inclusive—is the chance of a coin toss landing heads. Now, the sentence $C\frac{1}{2}$ specifies a theory of chance (that the coin tosses are fair), and thus, by the principal principle, $\left[\!\!\left[C\frac{1}{2} \vdash Rc\right]\!\!\right] = \frac{1}{2}$.

It seems to me that, unlike the axioms of probability, the principal principle is not a constraint on rationality. Rather, it is a constraint on chance: if something purports to be a propensity, but applying the principal principle on it is inappropriate, then that something cannot be a propensity. Lewis (1980) seems to agree, though he cautions that the principle is not an analysis of chance; but of course, a constraint does not have to be an analysis.

4.4 Dealing with first-order quantifiers

In science, one often encounters quantified statements: *all* humans are mammals, *all* electrons are smaller than the eye can see, and so on. The so-far developed machinery cannot express arguments containing such quantifiers. While one can express all kinds of problems in this formalism by just using a propositional language and informal explanations of the propositional constants' content (many expositions of Bayesian probability do not even discuss quantifiers, see, e. g., Jeffrey 2004; Howson and Urbach 2006), the ability to handle quantifiers adds to the ability to express subleties in the language, and to its realism.

The standard first-order predicate language $\mathcal{L}^1$ is obtained from $\mathcal{L}$ by adding two new kinds of formulas, universal $\forall xp$ and existential $\exists xp$ quantification. The interpretations of the language are the interpretations of $\mathcal{L}$, and the definition of truth is extended to cover these new formulas in the standard way. The notation $p[t/x]$ stands for a formula obtained from p by replacing every free occurrence of x with t, taking care not to capture any variables. This language is first order in that the variables may stand in for individuals of the interpretation but cannot be used in place of function or predicate symbols. All the definitions and results stated above for $\mathcal{L}$, except for the definition of a finite language, carries over to $\mathcal{L}_1$ unchanged; $\mathcal{L}_1$ is never a finite language.

Let me define that an individual i of an interpretation M is *nameable*, if there is a variable-free term t for which $[\![t]\!]_{\mathcal{M}} = i$ holds; the term t I will call a *name* of i. Interpretations whose individuals are all nameable I will call *fully nameable*.

Now, consider only fully nameable interpretations containing at most $n \in \mathbb{N}$ individuals; let $\{t_1, \ldots, t_n\}$ be a finite set of terms that contains names for all the individuals of the interpretation. In this situation, the quantifiers are redundant and can be eliminated in a very simple manner: $\forall xp$ becomes $p[t_1/x] \wedge \cdots \wedge p[t_n/x]$ and $\exists xp$ becomes $p[t_1/x] \vee \cdots \vee p[t_n/x]$. This is not a very useful translation because it severely limits the class of interpretations that are applicable (since, to be a true elimination of quantifiers, the translation must be carried out once for all interpretations).

The restriction to nameable interpretations is not severe. Let us have a language and an interpretation of that language which is not fully nameable. Add to the language new constants for each individual that is not nameable, and modify the interpretation to assign these constants to those individuals. This modification does not change the truth value of any of the original language's formulas.

The real problem is, instead, in the limit of the interpretation's size. Extend now the infinitary language $\mathcal{L}_\omega$ with the first-order quantifiers $\forall$ and $\exists$; I will name that language $\mathcal{L}^1_\omega$ (in the terminology of Karp 1965, this is $L_{\omega_1\omega}$). Define $\mathcal{H}$ as the set of all variable-free terms of the language. Now, the quantifiers can be eliminated by translating $\forall xp$ and $\exists xp$ to

$$\bigwedge_{t\in\mathcal{H}} p[t/x] \quad \text{and} \quad \bigvee_{t\in\mathcal{H}} p[t/x],$$

respectively; these translations preserve meaning in all fully nameable interpre-

tations containing finitely or countably infinitely many individuals.

Thus, any sentence of the first-order predicate language may be translated without change in its meaning to the quantifier-free infinitary language $\mathcal{L}_\omega$. From there, the machinery of the full probability theory is available through the $\mathcal{T}$ translation. However, I believe (but have not proven) that no such translation is possible directly from the full first-order predicate language because there is no way to translate an arbitrary countable union back to the first-order predicate language. The disadvantage of this approach is that uncountable domains, such as the set of reals, is not accommodated; however, solutions such as allowing uncountable disjunctions and conjunctions are possible.

There are also equations that connect the strength of an argument involving a quantified conclusion to the strength of its quantifier-eliminated variant without resorting to an infinitary language. Gaifman (1964, p. 3) gives one, which Scott and Krauss (1966, p. 224) call the *Gaifman condition*. Another is offered by Earman (1992, p. 37, notation adjusted); his equation is noted by Gaifman and Snir (1982, p. 501) to be equivalent to the Gaifman condition so long as attention is restricted to fully nameable countable models. The details are beyond the scope of this dissertation.

4.5 Inference from empirical data

Consider the following inductive argument: "This random swan is white, therefore all swans are white." Formally, let W be a unary predicate symbol for whiteness, let S be a unary predicate symbol for its argument being a swan, and let R be a unary predicate symbol for its argument being chosen randomly among all swan; further, let c be a constant symbol for "this swan". Then, the argument becomes $\forall x(Rx \to Sx), Rc, Wc \vdash \forall x(Sx \to Wx)$. Call $\forall x(Rx \to Sx), Rc$ the *background knowledge* K,[8] call Wc the *data* d, and call $\forall x(Sx \to Wx)$ the *hypothesis* h; this makes the argument $K, d \vdash h$. For the purposes of analyzing this argument, define the *elementary hypothesis* $h_{k\ell}$ for any nonnegative integers k and ℓ to be the sentence stating that there are exactly k white and ℓ non-white swans (given a binary predicate symbol for individual identity, this can be written as a finite sentence, though it is rather tedious and thus is omitted here). Note that these sentences are all pairwise incompatible. Then, supposing that there are only a finite number of swan, the *primary* hypothesis h is deductively equivalent to the infinite sentence

$$\bigvee_{k \in \mathbb{N}} h_{k0}$$

[8] Arguably, I should leave K partially unspecified, to allow for unstated and un-thought-of assumptions that bear upon the argument; see, e. g., Wenmackers and Romeijn (2015, first paragraph of Sec. 4.3). For simplicity, I will ignore that subtlety.

and the *alternative hypothesis* $\neg h$ is deductively equivalent to

$$\bigvee_{k=0}^{\infty} \bigvee_{\ell=1}^{\infty} h_{k\ell}.$$

Note that

$$\bigvee_{k=0}^{\infty} \bigvee_{\ell=0}^{\infty} h_{k\ell}$$

is true in all interpretations where there are a finite number of swans. Due to Bayes' theorem, the equation

$$[\![K, d \vdash h]\!] = \frac{\sum_{k=0}^{\infty} [\![K \vdash h_{k0}]\!] [\![K, h_{k0} \vdash d]\!]}{\sum_{k=0}^{\infty} \sum_{\ell=0}^{\infty} [\![K \vdash h_{k\ell}]\!] [\![K, h_{k\ell} \vdash d]\!]} \tag{36}$$

holds so long as the denominator is nonzero. The *priors* $[\![K \vdash h_{k\ell}]\!]$ of the hypotheses you may select freely subject to the constraint that

$$\sum_{k=0}^{\infty} \sum_{\ell=0}^{\infty} [\![K \vdash h_{k\ell}]\!] = 1.$$

The *likelihoods* $[\![K, h_{k\ell} \vdash d]\!]$ of the hypotheses given the data follow, via the principal principle (discussed on page 69), from the premise that c was randomly selected among all swan; thus $[\![K, h_{k\ell} \vdash d]\!] = \frac{k}{k+\ell}$, and Equation (36) simplifies to

$$[\![K, d \vdash h]\!] = \frac{\sum_{k=0}^{\infty} [\![K \vdash h_{k0}]\!]}{\sum_{k=0}^{\infty} \sum_{\ell=0}^{\infty} \frac{k}{k+\ell} [\![K \vdash h_{k\ell}]\!]} \tag{37}$$

This equation specifies the *posterior* of the primary hypothesis $[\![K, d \vdash h]\!]$ as a function of the *priors* $[\![K \vdash h_{k\ell}]\!]$ of the elementary hypotheses. Further computation would require specifying the prior, which is a subjective matter. However, note that the posterior function specified by Equation (37) is perfectly objective; one might imagine programming a small program that allows the user to specify their personal priors for the elementary hypotheses and computes their personal posterior given this data. Alternatively, one might explore this function to see what sort of priors are required to have the posterior reach a given threshold level. These examinations are beyond the scope of this dissertation.

While this example specified a single datum, the same approach is used to deal with any amount of empirical data, or statistics derived from the data. The techniques required are standard under the rubrik of Bayesian statistics (see, e. g., Lee 2012; Gelman et al. 2013; Gill 2015).

A more difficult case is one where one does not have the raw data but just a report of a common hypothesis test. Recall the theory of hypothesis testing introduced by Neyman and Pearson (1933) as an alternative to the Bayesian approach.

They proposed "rules to govern our behaviour with regard" (p. 291) to hypotheses, producing an acceptance or rejection decision for each hypothesis submitted for testing. In basic outline, they required the specification of a two hypotheses that exhaust all possibilities: the hypothesis to be tested h_0 and the alternative hypothesis h_1; note that $h_0 \vee h_1$ must be true in all possible worlds. They further required the specification of a required chance of rejecting a true h_0. The hypothesis test then proceeds by selecting a testing criterion that minimizes the chance of accepting h_0 when it is false. In current parlance, the chance of rejecting a true h_0 is termed the α-level, with the chance of accepting h_0 when the alternative hypothesis is true being known as the β-level (alternatively, one may talk about the chance of a Type I and Type II error, respectively).

Suppose the hypotheses h_0 and h_1 of a Neyman–Pearson hypothesis test are written in some first-order predicate language without quantifiers; since the error chances are prespecified in Neyman–Pearson testing, they do not depend on the data and therefore the data need not be expressible in the language. One can then add to this language the modal operators A and R for acceptance and rejection, respectively (see Gomolińska 1997, 1998). Treating modal sentences as predicate symbols with no internal structure, since my underlying theory is not equipped to handle modal operators, the error chances are the propensities $\alpha = P(\mathsf{R}h_0 \mid h_0)$ and $\beta = P(\mathsf{A}h_0 \mid h_1)$ for all h_0 and h_1 suitable for Neyman–Pearson testing.

A study that uses a hypothesis test to argue its point typically reports the α-level, and thus it is possible to use the principal principle to assign $[\![K, h_0 \vdash \mathsf{R}h_0]\!] = \alpha$. The same does not make much sense for β. Typically, the hypotheses under consideration have an underlying structure: there is a parameter space Θ, every point θ of which corresponds to a simple hypothesis h_θ mutually incompatible with all other simple hypotheses, and the parameter space is partitioned into the disjoint sets Θ_0 and Θ_1 such that

$$h_0 \Leftrightarrow \bigvee_{\theta \in \Theta_0} h_\theta \quad \text{and} \quad h_1 \Leftrightarrow \bigvee_{\theta \in \Theta_1} h_\theta$$

hold.[9] Now, since only one of h_θ can be true, $\beta = P(\mathsf{A}h_0 \mid h_1) = P(\mathsf{A}h_0 \mid h_\theta \wedge \theta \in \Theta_1)$ for that single $\theta \in \Theta$ for which h_θ is true. As P is viewed as an objective chance, β is a function of θ; and since in practice it is impossible to know which h_θ is true, β is also unknowable. One might think, $1 - \beta$ being called the power of a test, that conventional power analysis (see, e. g., Cohen 1988) might be of service. However, all it does is pick the θ arbitrarily (or based on the data under analysis, which is worse as discussed by, e. g., Hoenig and Heisey 2001).

Accordingly, it really does not make any sense to lift β directly via the principal principle. However, the function $\beta_\theta = P(\mathsf{R}h_0 \mid h_\theta)$, which is called the *power function* of the test (see, e. g., Neyman and Pearson 1938/1967; Neyman 1942), is typically known; thus, it is possible to lift each value of β_θ independently; thus,

[9] The machinery developed in this chapter can deal with this if Θ is finite or countably infinite. An uncountable Θ, such as $\mathbb{R}$, requires a development of the theory of random variables, which would add too much technical weight to this dissertation compared to the benefit to the strength of my argument.

we can set $[\![K, h_\theta \vdash \mathsf{R}h_0]\!] = \beta_\theta$. Note that the power function depends on α: if Θ_0 contains only a single value θ_0, then $\beta_{\theta_0} = P(\mathsf{R}h_0 \mid h_{\theta_0}) = P(\mathsf{R}h_0 \mid h_0) = \alpha$.

Now, in the case of a typical hypothesis test in scientific practice, the primary hypothesis is the null hypothesis h_0 of no effect and the alternative hypothesis h_1 postulates the existence of an effect, no matter how minuscule. The decision is to accept or reject h_0, the latter of which is taken to support h_1. Reckoning after the decision has been made (so that $\mathsf{R}h_0$ is equivalent to $\neg\mathsf{A}h_0$) and supposing that the divisors are nonzero, the argument strength is

$$
\begin{aligned}
[\![K, \mathsf{R}h_0 \vdash h_1]\!] = \frac{[\![K \vdash \mathsf{R}h_0 \wedge h_1]\!]}{[\![K \vdash \mathsf{R}h_0]\!]} &= \frac{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_1} h_\theta \right]\!\!\right]}{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta} h_\theta \right]\!\!\right]} \\
&= \frac{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_1} h_\theta \right]\!\!\right]}{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_0} h_\theta \right]\!\!\right] + \left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_1} h_\theta \right]\!\!\right]} \\
&= \frac{1}{1 + \dfrac{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_0} h_\theta \right]\!\!\right]}{\left[\!\!\left[K \vdash \mathsf{R}h_0 \wedge \bigvee_{\theta\in\Theta_1} h_\theta \right]\!\!\right]}} = \frac{1}{1 + \dfrac{\left[\!\!\left[K \vdash \bigvee_{\theta\in\Theta_0} (\mathsf{R}h_0 \wedge h_\theta) \right]\!\!\right]}{\left[\!\!\left[K \vdash \bigvee_{\theta\in\Theta_1} (\mathsf{R}h_0 \wedge h_\theta) \right]\!\!\right]}} \\
&= \frac{1}{1 + \dfrac{\sum_{\theta\in\Theta_0} [\![K \vdash \mathsf{R}h_0 \wedge h_\theta]\!]}{\sum_{\theta\in\Theta_1} [\![K \vdash \mathsf{R}h_0 \wedge h_\theta]\!]}} = \frac{1}{1 + \dfrac{\sum_{\theta\in\Theta_0} \beta_\theta\, [\![K \vdash h_\theta]\!]}{\sum_{\theta\in\Theta_1} \beta_\theta\, [\![K \vdash h_\theta]\!]}}
\end{aligned}
\tag{38}
$$

The most immediate consequence of this result (which is a special case of Bayes' theorem) is that the α-level alone (which is the only quantity usually reported) does not determine the argument strength. Simple eye-balling a reported test can therefore lead quite a bit astray. But even if the power function is known, there is more data required: the priors. Based on a similar argument along with several other arguments involving systematic biases in the commonly used research and publication process, Ioannidis (2005) famously concluded that "most claimed research findings [based on hypothesis testing] are false"; and in the context of screening testing, a similar argument exposes the *base rate fallacy* of assuming that the probability of a patient with a positive test result actually having the condition tested for is determined by the test's sensitivity and specificity alone (see, e. g., Meehl and Rosen 1955; Bar-Hillel 1980).

4.6 Overcoming subjectivity

The subjectivity of this framework presents a problem for the scientist. If an argument's strength depends on its recipient to the extent that any value between 0 and 1 can sometimes be assigned freely to the same argument, how can a scientist argue anything in such a way as to convince all rational persons?

A mathematical result of Gaifman and Snir (1982, § 2) is informative here. They consider a theory of probability defined over sentences like I do. Unlike my formulation, they have first-order integer arithmetic be rigidly designated, but this is the only significant diffrerence between the underlying theories. They define that a sentence $p \in \mathcal{S}$ *separates* two interpretations $\mathcal{M}_1$ and $\mathcal{M}_2$ if either $\mathcal{M}_1 \vDash p$ and $\mathcal{M}_2 \nvDash p$ or $\mathcal{M}_1 \nvDash p$ and $\mathcal{M}_2 \vDash p$ hold. They also define that a set of sentences $\Phi \subseteq \mathcal{S}$ is *separating* if for any two interpretations that are separated by some sentence there is a sentence in Φ that separates them. They define further, for any sentence $p \in \mathcal{S}$ and $\mathcal{M}$, the sentence

$$p^{\mathcal{M}} = \begin{cases} p & \text{if } \mathcal{M} \vDash p, \\ \neg p & \text{if } \mathcal{M} \nvDash p. \end{cases}$$

In other words, if p is a testable hypothesis, then by testing that hypothesis one learns $p^{\mathcal{M}}$ where $\mathcal{M}$ is the real world. Now, I can state Theorem 2.1 of Gaifman and Snir (1982):

Proposition 7. *Let P be a probability measure defined over sets of interpretations (as before, translation from sentences via $\mathcal{T}$ is left implicit). Let $p_1, p_2, \ldots$ be a separating countable sequence of sentences.*

1. *Let q be any sentence. Then, for almost all interpretations $\mathcal{M}$,*

$$P\left(q \,\middle|\, \bigwedge_{i=1}^{n} p_i^{\mathcal{M}}\right) \xrightarrow{n\to\infty} \begin{cases} 1 & \text{if } \mathcal{M} \vDash q, \\ 0 & \text{if } \mathcal{M} \nvDash q. \end{cases}$$

   holds.

2. *Let P' be another probability measure defined over sets of interpretations such that $P'(A) = 0 \Leftrightarrow P(A) = 0$ for all $A \subseteq \mathbb{M}$. Then, for almost all interpretations $\mathcal{M}$,*

$$\sup_{q\in\mathcal{S}} \left| P'\left(q \,\middle|\, \bigwedge_{i=1}^{n} p_i^{\mathcal{M}}\right) - P\left(q \,\middle|\, \bigwedge_{i=0}^{n} p_i^{\mathcal{M}}\right) \right| \xrightarrow{n\to\infty} 0$$

   holds.

In effect, the first part of the theorem states that for any rational person and any theory, there is almost certainly sufficient conceivable empirical evidence to effectively convince them of the theory's truth value; the second part states that for any two rational persons that are equally dogmatic (that is, they assign zero probability to the same sets of interpretations), there is almost certainly sufficient

conceivable empirical evidence to make them effectively agree on everything. Whether such evidence can be practically generated is another matter entirely; while that is certainly an important problem in practice, it is also beyond the scope of any general theory of empirical reasoning. In terms of argument evaluation, this theorem states that given enough premises, individual freedom of choice becomes almost surely irrelevant.

It initially appears to follow from this theorem that a scientist is always better off listing more premises in support of their conclusion. Indeed, the theorem does show almost sure convergence of argument stength as the premise list grows. However, a strong argument is only a part of what a scientist must achieve, as an argument is ultimately always hypothetical: *if* you, dear reader, believe my premises, *then* your credence for my conclusion ought to be equal to the strength of my argument as you have evaluated it. To be effective, therefore, the argument ought to list premises that are easier to believe than the conclusion would be on its own. This is why an empirical argument, where the premise is a conjunction or summary of observation sentences,[10] is one of the best inductive arguments that can be had; but other kinds of inductive arguments can be of use as well.[11]

Another approach is to frame the scientist's goal as inducing consensus. In other words, it is rough intersubject agreement that is sought after. Thus, instead of considering a single argument strength function, one now considers a collection of them, selected as representing the range of relevant individual choices. A similar approach within Bayesianism has been advocated by, e. g., Joyce (2005, 2010) and Benétreau-Dupin (2015) under the label *imprecise credences*, where a rational person's beliefs are modeled not by a single credence but a set of them. In the case of argument evaluation, it is tempting to state that the more the various argument strength functions agree about the strength value, the stronger the argument is *objectively* (cf. Joyce 2005). Or perhaps, instead of computing argument strength from the priors, one should choose a threshold strength and solve the resulting inequality for the priors, to determine the level of prior disbelief required to dispute the argument (cf. Dawid 1987).

---

10 Quine (1993, p. 108) characterizes *observation sentences* as follows: "They should be sentences like 'It's cold', 'It's raining', 'That's milk', 'That's a dog', to which we have learned to assent unreflectively on the spot if we are queried when certain associated sensory receptors are triggered. [...T]ake the sentence 'There is some copper in the solution'. [...I]t becomes an observation sentence for a chemist who has learned to spot the presence of copper by a glance at a solution."

11 Andreas Stefik points out (in his comments attached to his review of this dissertation) that this seems "a very complex way of stating what is rather obvious from history". Very true. This section addresses one of the biggest potential stumbling blocks of this theory of induction, and being able to derive a commonsense result should be viewed as a falsification attempt survived.

4.7 Evidence

"What's your evidence?"

If you ask me this question, it is, I'm sure, because I had made a claim that you doubt is true. However, I doubt you would ask me this if I had claimed that there are infinitely many primes p for which $p + 2$ is also prime (McKee 2013); you would have, instead, requested to see my proof. Requesting evidence seems to make sense only if the claim is contingent, that is, the claim is about the real world, not about all possible worlds.

For concreteness, suppose I assert that static typing is a better choice for a general-purpose programming language than dynamic typing in that it reduces actual programmer error rates. Disregard for the purposes of this thought-experiment the problem of vagueness of this claim. When you are asking for my evidence, you are, I believe, asking me to supply premises to an argument whose conclusion is my claim.

Suppose I answer, "I read it on a mailing list sometime in the late 1990s." You would feel, I believe, unsatisfied. Evidence must be well defined, so that it can be scrutinized.

Or, I could say, "Here is a blog post by a blogger I admire, asserting the same claim I made." Suppose I actually point you to such a blog post. You read the post and find it to indeed assert the claim but to provide no argument in support. You are unsatisfied. Do you reply, "That evidence is not convincing for me"? I doubt it. You would, I believe, respond, "That's not evidence!"

Then, I might supply the following. "Static typing ensures that no type errors happen at run-time." You might agree that this is true by definition, but you would likely feel uneasy. How does a statement that is deductively true shed light on a contingent claim?

Suppose that the blog post I pointed to you did offer an argument, but it is essentially the same argument I discussed in the previous paragraph. This would, I believe, not help my case at all. It is still a deductively true statement that does not give any help to a contingent claim.

Then, I might tell you of an experiment I conducted,[12] where I took fifty programmers, each experienced with both dynamically and statically typed languages, randomized them to four groups, had two groups develop a particular program in a statically typed language (different for each group) and two groups develop that same program in a dynamically typed language (different for each group). I further tell you that the statically typed groups had statistically significantly fewer bugs than the groups using dynamically typed languages. I believe you would concede that this is evidence, as I am reporting actual observations I made, though whether you find it convincing is another matter entirely.

Or, I might refer you to published reports about such experiments conducted by others. I believe you would accept it as evidence, though not necessarily convincing.

[12] To be clear, I have not actually conducted this experiment regarding type systems.

Suppose I had in fact conducted the experiment I mentioned above but did not tell you about it. Would you still consider it evidence? Certainly not. You are not even aware of it; how could you consider it evidence?

Therefore:

Analysis 8. Evidence *is a report of observations offered as a premise in an argument whose conclusion is contingent, provided that the presence of this premise affects the strength of the argument.*

This analysis would make no sense if the only argument that one would accept is a valid (i. e. deductive) one. Thus, this analysis depends on an analysis of inductive argument, such as that I developed earlier in this chapter. The proviso requiring an effect on argument strength speaks about relevance of the evidence. It should be noted that this analysis does not pay any attention in the weight of the evidence (cf., e. g., Joyce 2005).

In my Licentiate Thesis (Kaijanaho 2014), I phrased this analysis in a somewhat different way, and offered it as a definition. I repeat the definition here, because it is relevant to the mapping study reported in later chapters.

Definition 9 (For the purposes of the mapping study)**.** 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.*

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.*

This definition is based on a Bayesian approach; instead of considering inductive arguments, I considered the effect of the evidence on a Bayesian agent, that is, an agent whose actions are governed by a credence function that obeys the axioms of probability. The resulting definition is nearly identical in substance to Analysis 8.

The definition, being for the purposes of the mapping study, also defines scientific evidence, and provides examples of the kind of material that is excluded. In my view, this definition is obvious and does not require argument, although it is probably possible to refine it via good arguments; such refinements are, however, beyond my scope here.

4.8 Related approaches

The label "Bayesian" is applied to a large number of distinct bodies of research. Common to them all is the use of probability theory and the formulation of problems in a way that makes the use of Bayes' theorem prominent in their solution.

I wish to here briefly comment on several of them, and thus point out that the theory of induction I have argued for is distinct from them.

Bayesian confirmation theory (see, e. g., Hawthorne 2014) is the closest. Its roots are in the logical empiricist program of explicating a formal system that allows one to determine whether a body of empirical evidence objectively confirms a scientific theory and, if so, how much (see, e. g., Hempel 1945); the Bayesian approach assumes a Bayesian logic of induction or a theory of Bayesian agents and uses that apparatus to investigate the problems of confirmation. As such, it basically starts where I end; the theory of confirmation is not germane to this dissertation.

Also clearly related is Bayesian statistics (see, e. g., Lee 2012; Gelman et al. 2013; Gill 2015), where statistical data are analyzed by deriving from them a likelihood function which is then combined with a prior distribution using Bayes' theorem to yield a posterior distribution, and the data are summarized via various visual and numeric presentations of the posterior distribution. As Berger (2006, p. 464) points out, the goals of statistics and a theory of reasoning are different, and they thus make quite different choices; in statistics, for example, subjectivity is much more problematic than it is in a theory of induction, and the standard solution of statistics—of using special theoretical priors—makes it difficult to apply a published Bayesian analysis of statistics in a Bayesian inferential framework.

Somewhat removed from my approach is the use of probability theory in machine learning and artificial intelligence (see, e. g., Ghahramani 2015): such techniques include Bayesian classifiers and Bayesian networks. The focus there is, like in all computer science, the efficient implementation of algorithms, which is a consideration not germane to a foundational theory like the theory of induction.

4.9 Discussion

In this chapter, I have argued (i) that the strength of an (inductive) argument can be measured by a function that behaves like conditional probability, and therefore the full theory of probability can be brought to bear; (ii) that the strength of an inductive argument is a subjective notion, and depends on the person doing the evaluation; (iii) that, nevertheless, some objectivity can be reclaimed by abstracting over the class of likely evaluators; and (iv) evidence is a premise in an inductive argument.

All of these arguments are fundamentally inductive themselves. Consider for example the argument that inductive strength is probability: There is no reason why elements of V have to be real numbers, and further there is no reason why $(V, \leq)$ must be totally ordered. There were alternative choices that were available but not taken, mainly because this allowed the development of the theory to the direction needed to identify it with the standard probability theory. Similar choices are available in the explication of probability (cf. Fine 1973) and were not taken by the people who developed or accepted the standard theory. In

addition, a number of equations were generalized away from the domain where they were deduced to the full set V: since the standard theory of probability is known to be no more inconsistent than the standard foundations of mathematics, this simplification does not cause contradictions, but it does make the argument vulnerable to criticism like that of Halpern (1999).

The adoption of an infinitary language and countable additivity (i. e. Equation 35) were purely for the convenience of achieving identity with the infinite part of the probability theory, with no better justification offered. However, the theory of probability itself has the same problem. Kolmogorov (1933/1956) describes the adoption of an axiom equivalent to countable additivity as follows (p. 15, emphasis deleted):

> "We limit ourselves, arbitrarily, to only those models that satisfy Axiom VI. This limitation has been found expedient in researches of the most diverse sort."

He further, on pages 16–18, justified requiring σ-algebras by noting that any algebra may be extended to be a σ-algebra, and the assumption of a σ-algebra allows the development of the theory without having to deal with possible nonexistence of countable unions. He acknowledged that this extension creates a mathematical structure which may no longer have any empirical relevance but that using it to derive empirically relevant results guarantees that those results are noncontradictory. In other words, this whole machinery of infinite probability is a mere convenience with no stronger justification. The recent textbooks on probability theory which I have consulted appear to agree: Ash (1970/2008) treats countable unions and countable sums as obvious, unproblematic generalizations of finite ones; Billingsley (1995, p. 18) motivates the use of infinite σ-algebras as them being as large as is possible without introducing technical difficulties; Pollard (2002) regards infinite σ-algebras and countable additivity with suspicion but indispensable for a general theory; and Durrett (2013) defines a probability space directly as a σ-algebra without comment.

One (like Popper 1935/2002, Sec. 1.1) might argue that in arguing for induction inductively, I have committed the philosopher's cardinal sin of *petitio principii*, that is, taking my conclusion as a premise. But, what choice do I have? I firmly agree with Hume (1740/2003, p. 64, emphasis deleted)—

> "there can be no [deductive] arguments to prove, that those instances, of which we have had no experience, resemble those, of which we have had experience"

—and with Popper (1935/2002, Sec. 1.1)—

> "A principle of induction would be a statement with the help of which we could put inductive inferences into a logically acceptable form. [...] Now this principle of induction cannot be a purely logical truth like a tautology or an analytic statement."

—that induction cannot be justified with a deductive argument. Since, in my view, the only other kind of argument is an inductive one, it is an inductive argument I must offer.

Popper (1935/2002, Sec. 1.1) basically argues that induction cannot be supported without induction, therefore induction is to be rejected. But if this is to be

taken as a deductive argument (and I cannot see how Popper could coherently use induction to reject induction), there must be a hidden premise. So far as I can tell, that hidden premise is a distrust of induction. Thus, the Popperian argument becomes: induction is to be rejected; induction cannot be supported without induction; therefore, induction is to be rejected. If that is not an example of *petitio principii*, I do not know what is.

Of course, the very same *petitio* meta-arguments apply to deductive reasoning as well. Either one accepts deduction *a priori*, in which case one can deduce acceptance of deduction, or one doesn't, in which case one can't. Therefore, it will not save Popper's argument to say that the hidden premise is "an argument technique cannot be justified using that argument technique" (which is merely a statement disallowing *petitio principii*).

In my view, the correct reading of any argument for any argumentation scheme (inductive or deductive) is to see it as supporting a specific scheme. My inductive argument supports induction viewed as an instance of probability theory; it does not say anything about other possible ways of understanding induction. The standard argument deductively showing the soundness and completeness of first-order predicate logic (which is deductive) says nothing about other kinds of deductive logic. Viewed in this light, all charges of *petitio* must be dismissed.

To recap, the logic of induction offered in this chapter has a dual purpose in this dissertation. First, it provides a foundational theory to support the inductive arguments I will be making. Second, it provides a basis for building the evidence-based programming language design that I will undertake in Chapter 10, by providing a framework for evaluating the strength of empirical arguments.

From these foundational issues, I will move on to the practical methodology of secondary studies in the next chapter.

5 SYSTEMATIC SECONDARY STUDIES

This dissertation includes a report of 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. This chapter appeared earlier in my Licentiate Thesis (Kaijanaho 2014), but I have updated it lightly to take into account recent research.

5.1 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 or a representative sample, 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 da Silva, Santos, Soares, França, and Monteiro 2010; Santos and da 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, Turner, et al. 2009; Kitchenham, Pretorius, et al. 2010; da 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 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).

5.2 Best-practice methodology

In the following sections, 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. Further, 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. Finally, Petersen, Vakkalank, et al. (2015) have proposed updates on mapping study guidelines based on a systematic map of systematic maps.

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.

5.3 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).

5.4 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.

5.5 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.

Petersen, Vakkalank, et al. (2015, p. 10) note that a systematic mapping study does not always aim to locate all relevant studies, just a representative sample. Thus, they recommed that only a subset of all available search modalities be used in a mapping study.

5.6 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.

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).

5.7 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.

Petersen, Vakkalank, et al. (2015) recommend that mapping studies standardize topic classifications where possible. For classifying publication venues, they recommend using the Finnish *Ministry of Education Publication Type Classification* (2010).

5.8 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.

Petersen, Vakkalank, et al. (2015) recommend that mapping studies discuss threats to their validity. They list the following possible threats: publication bias, poor design of data extraction, and researcher bias. They further note that sample quality, generalizability, and the reliability of conclusions are relevant to validity.

I have now summarized the current recommended practice for systematic secondary studies primarily in software engineering. These recommendations influenced the design of the mapping study in this dissertation. 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. In the next chapter I will examine one issue of secondary study methodology more thoroughly, since it seems to me the current recommendations do not address it critically.

6 CODING RELIABILITY ANALYSIS

Coding reliability analysis is a necessary part of a systematic secondary study; as previously mentioned, Kitchenham and Charters (2007, p. 20) recommend it for evaluating selection decisions, but it is equally useful in evaluating data extraction. However, in software engineering secondary study methodological literature, guidance about the correct method for conducting these analyses is sparse. Therefore, in this section, I will review key results related to Cohen's kappa and related statistics. Since the situations where coding reliability is an issue in secondary studies are effectively exercises in content analysis, this review is heavily influenced by the basic framework of coding reliability in content analysis (see particularly Krippendorff 2013, Chapter 12).

Situations where coding reliability becomes an issue are generally of the following type. A study is interested in determining the value of a particular attribute of objects of interest, and it goes about this by designing a *coding scheme* and then having one or more human beings (*coders*, *raters* or *judges*; generally, *observers*) apply the scheme to a set of objects (*units* or *individuals*) in order to assign a single *value* to each. This process is called *coding*. Typically in these situations, there is no gold standard to compare to, and thus there is no way to compare the resulting values to "true" values. Often it is not even clear if the attribute of interest exists or can be coherently created.

Krippendorff (2013, p. 270–272) distinguishes between three facets of reliability in this context. First is *stability*: if the same coding scheme is applied to the same units by the same coder at different times, how identical are the results? This is a question about *intra-coder agreement*, and is answered by a test–retest procedure. Second is *reproducibility*: if the same coding scheme is applied to the same units by different coders working independently of each other, how identical are the results? This is an *inter-coder agreement* question, and is answered by a test–test procedure. Third is *accuracy*: does the attribute created by the coding process correspond with the real attribute it aims to mimic? This is a validity issue, and often is unanswerable because the real attribute, if it exists, may be unknowable.

Consider the mapping study reported in Chapters 7 and 8. The decision to include or exclude a particular article from the study required me, the researcher,

to read the article and decide, based on defined criteria (see Subsection 7.3.1), whether the article is to be included or excluded. For example, I looked at a particular article, noticed it is written completely in Chinese, recalled that one of the exclusion criteria specifies the allowed languages, and thus entered an exclusion decision. On another article, I had to think for hours whether the article studies a language design decision or not. The former decision is easy, but the latter is not, and it is quite possible that another researcher would disagree. Thus, I submitted a random sample of the articles to other researchers, who made their own decisions. In other words, I conducted a test-test procedure to evaluate inter-coder agreement: when we actually disagree, how serious is our disagreement, and does chance explain the extent of our agreement? This is what coding reliability analysis is about.

Formally, let U be the (non-empty) finite set of units and V be the (non-empty) set of values. Then, the attribute of units elicited by a coding scheme can be modeled as a random function[1] C from U to V. Successive *observations* of a unit $u \in U$ generate a sequence of values $c_{u,1}, c_{u,2}, \ldots, c_{u,i}, \ldots \in V$. If one is interested in intra-coder agreement, the sequence consists of successive values generated by the same coder; but one can equally well be interested in inter-coder agreement, in which case every value in the same unit's sequence is generated by a different coder.

There is a large number of proposed statistics of inter-coder agreement, several of which I will be discussing in this chapter. Generally, they all take a sample from C and yield a real number whose normal range is $[0, 1]$ but which may in some circumstances be negative, with greater values indicating greater inter-coder agreement. Some of them are also used for measuring intra-coder agreement.

Proofs of numbered propositions in this chapter are given in Appendix 1.

6.1 Two nominal observations per unit

Let us first consider the case where every unit $u_i \in U = \{u_1, \ldots, u_n\}$ receives exactly two observations, $c_{i,1}, c_{i,2} \in V$. In this case, the sample of C is typically characterized by a *contingency table*

	v_1	$\cdots$	v_m	
v_1	n_{11}	$\cdots$	n_{1m}	$n_{1\cdot}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
v_m	n_{m1}	$\cdots$	n_{mm}	$n_{m\cdot}$
	$n_{\cdot 1}$	$\cdots$	$n_{\cdot m}$	$n_{\cdot\cdot}$

where each n_{ij} is the count of units whose first observation is v_i and the second is v_j, and the marginals are the obvious sums. From this table, the following

[1] That is, a random variable whose values are functions. As usual, the word "random" does not here imply that a chance process must be involved.

fractions can be derived:

$$p_k = \frac{n_{\cdot k}}{n_{\cdot\cdot}}, \qquad q_k = \frac{n_{k\cdot}}{n_{\cdot\cdot}}, \qquad r_k = \frac{p_k + q_k}{2},$$

which represent the fraction of the units that received the v_k value as their first, second, and any observation, respectively.

Let us further assume that there are finitely many values $V = \{v_1, \ldots, v_m\}$ for which any single disagreement is equally serious (i. e. the observations are *nominal*). Now, the *proportion of (observed) agreement*, also called the *raw agreement*, is

$$P_o = \frac{1}{n_{\cdot\cdot}} \sum_{i=1}^{m} n_{ii}.$$

However, since even drawing values out of a hat will result in some agreement, the raw-agreement statistic is not very informative. If one can determine the proportion of agreement P_e that would be expected under those circumstances, one could define a chance-corrected statistic of agreement as the ratio of actual agreement beyond chance to the greatest possible agreement beyond chance:

$$\frac{P_o - P_e}{1 - P_e}.$$

The chance-corrected agreement lies within $]-1, 1]$. It is undefined if and only if $P_e = 1$; otherwise, it is 1 if and only if $P_o = 1$ and 0 if and only if $P_o = P_e$. Thus 1 indicates perfect agreement and 0 indicates that the observed agreement is indistinguishable from agreement purely by chance. Negative values are possible if there is systematic disagreement that outweighs agreement by chance.

Several definitions of P_e are available in the literature, of which I will discuss several that are mentioned in debates about coder agreement. I will be distinguishing between different versions of P_e by superscripting it with the symbol of the corresponding chance-corrected agreement statistic.

If we assume that the sample was generated by having two coders observe each unit independently and that one coder's observations are recorded as $c_{i,1}$ and the other coder's observations are recorded as $c_{i,2}$ (regardless of the temporal relationship between the observations), the proportion of the first coder's value assignments is p_i for each value v_i, and similarly q_i for the second coder. Now, the proportion of agreement excepted under the assumption that both coders are drawing values from their own hats, the contents of which are distributed according to p_k and q_k, respectively, is

$$P_e^{\kappa} = \sum_{k=1}^{m} p_k q_k, \tag{39}$$

from which Cohen's (1960) kappa, written in formulas by κ, arises as the corresponding chance-corrected measure of agreement.

Now, consider the situation where there may be any number of coders working together to generate two independent observations for each unit. As Krippendorff and Fleiss (1978) note, the assumption of two coders mentioned above

means that Cohen's kappa is inappropriate here. Instead, we may calculate the proportion of agreement expected under the assumption that each coder draws values (with replacement) from a hat common to all coders, the distribution of which is the observed distribution of values in the sample, that is, r_i for each value v_i. Since two values are drawn from the hat for each unit, the proportion of agreement expected under chance becomes

$$P_e^{\pi} = \sum_{k=1}^{m} {r_k}^2. \tag{40}$$

The corresponding statistic of chance-corrected argreement is Scott's (1955) pi, written in formulas as π.

It makes sense that the values drawn for one unit are replaced before another unit is considered, as they are obviously independent. But why should the drawing from the hat be with replacement inside a unit? Observations generated for a unit are generated by observing the *same* unit and therefore are arguably not independent, and it therefore makes some sense to model the chance agreement as being the result of drawing without replacement within units, but with replacement between units; thus,

$$P_e^{\alpha} = \sum_{i=1}^{m} \left(\frac{n_{.i} + n_{i.}}{2n_{..}} \times \frac{n_{.i} + n_{i.} - 1}{2n_{..} - 1} \right). \tag{41}$$

The resulting statistic of chance-corrected argreement is Krippendorff's (2011, 2013) alpha, written in formulas as α.

A phenomenon common to all three statistics was pointed out (in the case of Cohen's kappa) by Kraemer (1979), Grove et al. (1981), and Feinstein and Cicchetti (1990). Let us first define the following quantities:

$$\tilde{p}_k = p_k - \frac{1}{m}, \qquad \tilde{q}_k = q_k - \frac{1}{m}, \qquad \tilde{r}_k = r_k - \frac{1}{m}.$$

Thus, each of $\tilde{p}_k$, $\tilde{q}_k$, and r_k is the (directed) deviation of the corresponding value assignment proportion from what would be expected under a uniform distribution of values, and thus for its part quantifies the nonuniformness of the observed value distribution.

We may now express each of the various P_es in terms of the deviations just defined:

Proposition 10.

$$P_e^{\kappa} = \frac{1}{m} + \sum_{k=1}^{m} \tilde{p}_k \tilde{q}_k, \qquad P_e^{\pi} = \frac{1}{m} + \sum_{k=1}^{m} \tilde{r}_k^2,$$
$$P_e^{\alpha} = \frac{1}{2n_{..} - 1} \left(2n_{..} \sum_{k=1}^{m} \tilde{r}_k^2 + \frac{2n_{..} - m}{m} \right).$$

Notice that P_e^{π} and P_e^{α} increase (and thus π and α decrease) when the magnitude of the deviation, $\sum |\tilde{r}_k|$, increases. Similarly, when both coders deviate in the

same direction for each value, P_e^κ increases (and κ decreases) when the magnitude of either coder's deviation, $\sum|\tilde{p}_k|$ or $\sum|\tilde{q}_k|$, increases. Such deviation indicates that some values are much rarer than other values in the sample of observations; if the coders are accurate, this observerd deviation also indicates a true disparity in the prevalence of the values. Thus:

Remark 11. *If the coders are extremely (but not perfectly)*[2] *accurate, but some values are much rarer than others, then the value of π, α, and κ is low despite the coders' accuracy.*

This phenomenon was called "the first paradox of κ" by Feinstein and Cicchetti (1990, p. 544). Their "second paradox" is discussed below; see Remark 16 on page 100.

Grove et al. (1981) criticized Cohen's kappa (and, implicitly, also Scott's pi and Krippendorff's alpha) in the context of evaluating the reliability of psychiatric diagnosis for using a misleading model of chance agreement (p. 411):

> "'Chance agreement' means the agreement that would be observed if two raters assigned diagnoses to cases at random. Now this is not what diagnosticians do. They assign the easy cases, or 'textbook' cases, to diagnoses with little or no error; they may guess or diagnose randomly on the others."

Agreeing with Grove et al. (1981), Gwet (2008) posits another model of chance agreement: a coder either is sure (and correct) about what value to assign to a unit (a *textbook case*) or assigns the value by drawing it (with replacement) from a hat containing the same number of copies of each value (a *hard case*). The coders may not agree which units are textbook cases and which are hard cases, nor is an observer's determination of textbook-case status known. Gwet suggests that the proportion of hard case units within all units is estimable by seeing how much the observed multinomial distribution of observed values differs from an uniform multinomial distribution, measured by the ratio of observed variance to uniform variance for each possible value, averaged over the full set of possible values[3]

$$\begin{aligned}\Phi^{AC_1} &= \frac{1}{m}\sum_{k=1}^{m}\frac{r_k(1-r_k)}{(1/m)(1-1/m)} = \frac{1}{m}\sum_{k=1}^{m}\frac{r_k(1-r_k)}{1/m-1/m^2} = \frac{1}{m}\sum_{k=1}^{m}\frac{r_k(1-r_k)}{(m-1)/m^2}\\ &= \frac{1}{m}\sum_{k=1}^{m}\frac{m^2}{m-1}r_k(1-r_k) = \frac{m}{m-1}\sum_{k=1}^{m}r_k(1-r_k).\end{aligned}$$

Gwet further suggests that the proportion of hard case units for which random agreement can be expected is estimable by assuming that both observers agree on which units are hard cases, making it equal to the probability that two throws of an m-sided die result in both resulting in the same value, that is $1/m$. Given

---

2 If the coders are perfectly accurate, they agree without exception, and $\pi = \kappa = \alpha = 1$ (although one or more of them may be undefined).

3 Gwet (2012, p. 82) justifies the same formula by stating that it is the complement of the standardized chi-square distance between distributions applied to the observed distribution of values and to the corresponding uniform distribution, thereby quantifying the difference between the two distributions.

these assumptions, the proportion of units having both coders agree on a value and at least one of them having assigned that value randomly is

$$P_e^{AC_1} = \frac{1}{m}\Phi^{AC_1} = \frac{1}{m}\frac{m}{m-1}\sum_{k=1}^{m} r_k(1-r_k) = \frac{1}{m-1}\sum_{k=1}^{m} r_k(1-r_k). \qquad (42)$$

The corresponding statistic of chance-corrected argreement is Gwet's (2008, 2012) AC_1.

This statistic, too, can be reformulated based on the distance from a uniform distribution.

Proposition 12. $P_e^{AC_1} = \frac{1}{m} - \frac{1}{m-1}\sum_{k=1}^{m} \tilde{r}_k^2$.

Note that, unlike with the previous statistics, the chance agreement statistic decreases (and AC_1, correspondingly, increases) as the magnitude of deviation increases.

An alternative formulation of these statistics, favored by Krippendorff (1970, 2004, 2011, 2013) and more useful for later developments, is achieved by setting the following definitions:

$$D_o = 1 - P_o, \qquad D_e = 1 - P_e.$$

Since P_o and P_e range between 0 and 1 and are statistics of observed and expected agreement, respectively, so do D_o and D_e range between 0 and 1 and measure observed and expected *disagreement*, respectively. It is easy to see that, given a statistic of agreement expected under chance, $P_e \in [0, 1]$, the corresponding statistic of chance-corrected agreement is

$$1 - \frac{D_o}{D_e}.$$

Such statistics can therefore be seen as the complement of a corresponding statistic of chance-corrected *disagreement*, the latter being simply the ratio of observed disagreement to expected disagreement.

One problem of a contingency table is that the dimensions have different semantics, which will make generalization to more than two observations per unit difficult. To solve this issue, Krippendorff (2011, 2013) has suggested switching to what he calls a *coincidence table*:[4]

	v_1	$\cdots$	v_m	
v_1	o_{11}	$\cdots$	o_{1m}	$o_{1\cdot}$
$\vdots$	$\vdots$	$\ddots$	$\vdots$	$\vdots$
v_m	o_{m1}	$\cdots$	o_{mm}	$o_{m\cdot}$
	$o_{\cdot 1}$	$\cdots$	$o_{\cdot m}$	$o_{\cdot\cdot}$

[4] Krippendorff (1970, 2013) attributes the idea of coincidence tables to Pearson (1901), and indeed, on pages 292–293 Pearson did discuss the generation of tables of this type, calling them a kind of "correlation table". The influential textbook of Fisher (1925/2005) called this sort of table a "symmetrical table" (p. 178). Neither Pearson nor Fisher attributed the idea to anyone.

It is defined by $o_{k\ell} = o_{\ell k} = n_{k\ell} + n_{\ell k}$. Notice that the array is symmetric with respect to its diagonal. For simplicity, let us set the following further notational definition:

$$o_k = o_{k\cdot} = o_{\cdot k}.$$

The previously used proportion of use of a particular value v_k, written as r_k, has a natural expression in terms of the coincidence table:

$$r_k = \frac{o_k}{o_{\cdot\cdot}}$$

The key difference between a contingency table and a coincidence table is that the former partitions the observations into two sets according to their sequence index—the first set is $\{\, c_{i,1} \mid u_i \in U \,\}$ and the second is $\{\, c_{i,2} \mid u_i \in U \,\}$—and these sets form the two dimensions of the contingency table; whereas a coincidence table lumps all observations of a unit together regardless of their positions in the observation sequence, and thus the dimensions are indistinguishable. Scott's pi and Krippendorff's alpha have simple formulas in the contingency table reformulation, once two pieces of new notation are introduced:[5]

$$\delta_{ij} = \begin{cases} 1, & \text{if } i \neq j, \\ 0, & \text{if } i = j. \end{cases}, \quad r_{k\ell} = \frac{o_{k\ell}}{o_{\cdot\cdot}}.$$

Proposition 13.

$$D_o = \sum_{i=1}^{m}\sum_{j=1}^{m} r_{ij}\delta_{ij}, \qquad D_e^{\pi} = \sum_{i=1}^{m}\sum_{j=1}^{m} r_i r_j \delta_{ij},$$

$$D_e^{\alpha} = \frac{o_{\cdot\cdot}}{o_{\cdot\cdot} - 1}\sum_{i=1}^{m} r_i \sum_{j=1}^{m} r_j \delta_{ij} = \frac{o_{\cdot\cdot}}{o_{\cdot\cdot} - 1} D_e^{\pi},$$

$$P_o = \sum_{i=1}^{m} r_{ii}, \qquad P_e^{\alpha} = \frac{1}{o_{\cdot\cdot} - 1}\left(o_{\cdot\cdot} \sum_{i=1}^{m} {r_i}^2 - 1 \right).$$

The factor differentiating α and π from each other originates in the choice of how chance agreement is modeled: the former picks values inside a unit without replacement, the latter with replacement. Their numeric difference can be computed (this result has been stated without proof by Krippendorff 2004, p. 419):

Proposition 14. $\alpha - \pi = \dfrac{1 - \pi}{o_{\cdot\cdot}}$.

Note that, if $\pi < 1$, then $\pi < \alpha$. Thus, the difference between α and π diminishes when the sample becomes larger and vanishes at the limit.

The difference between κ and either π or α is not so simple to characterize, but the following is fairly simple to demonstrate (proved by e. g., Warrens 2010, Theorem 2):

---

5 The δ notation is due to Krippendorff (2011, 2013), simplified here a bit. Note that δ_{ij} is *not* the Kronecker delta but its exact opposite.

Proposition 15. *$\pi \leq \kappa$; and $\pi = \kappa$ if and only if $p_k = q_k$.*

The following phenomenon was called by Feinstein and Cicchetti (1990, p. 545) the "second paradox" of κ:

Remark 16. *When the disagreement between two coders about the distribution of values increases, κ also increases.*

An interesting identity, which Feng (2013, p. 2964) proved for the special case $m = 2$, exists between $P_e^{AC_1}$ and D_e^{π}, that is, Gwet's statistic of expected agreement and Scott's statistic of expected *dis*agreement!

Proposition 17. $P_e^{AC_1} = \dfrac{D_e^{\pi}}{m-1}$.

Another characterization, one I have not seen mentioned in the literature, of the difference between expected agreement in π and AC_1 can be obtained in terms of the deviation from uniform distribution of values:

Proposition 18. $P_e^{AC_1} = P_e^{\pi} - \dfrac{m}{m-1} \sum_{k=1}^{m} \tilde{r}_k^2$.

Now, one can determine the circumstances when each of the four statistics obtain one of the special values (undefined, 0, or 1).

Proposition 19. *κ is undefined if and only if there is some $i \in \{1, \ldots, m\}$ such that $p_i = q_i = 1$.*

Proposition 20. *α is undefined if and only if π is undefined. Further, they are undefined if and only if there is some $i \in \{0, \ldots, m\}$ such that $r_i = 1$.*

In other words, Cohen's kappa is undefined when both coders agree that there is no differentiation between the units and about the particular value must be applied to all units; and Krippendorff's alpha and Scott's pi are undefined when all observations agree that there is no differentiation and about the single value to be used. However:

Proposition 21. *AC_1 is always defined.*

Next, I will characterize when each of the statistics evaluate to zero.

Proposition 22. *$\kappa = 0$ if and only if*

$$\sum_{i=1}^{m} (r_{ii} - p_i q_i) = 0;$$

$\pi = 0$ if and only if

$$\sum_{i=1}^{m} \left(r_{ii} - {r_i}^2 \right) = 0;$$

$\alpha = 0$ if and only if

$$\sum_{i=1}^{m} \left(r_{ii} - \frac{o_{..}}{o_{..} - 1} {r_i}^2 \right) = \frac{1}{o_{..} - 1}.$$

This proposition has a natural interpretation in terms of the model of chance agreement that each of the statistics is based on. In each case (for α, only approximately), a zero statistic value indicates a zero cumulative (directional) difference between the observed agreement and what would be expected if values were being drawn from one or more hats, as appropriate. Since the difference is directional, equivalent deviations to opposite directions will result in a zero statistic value (cf. Remark 11). The following result is more puzzling:

Proposition 23. $AC_1 = 0$ *if and only if*

$$\sum_{i=1}^{m}\sum_{j=1}^{m}\left(r_{ii} - r_i r_j\right)\delta_{ij} = 0.$$

A special case highlights the puzzle:

Proposition 24. *If* $m = 2$, *then* $AC_1 = 0$ *if and only if*

$$r_1 r_2 = \frac{r_{11} + r_{22}}{2}.$$

Thus, in the binary special case, a zero AC_1 indicates that the average chance of agreement is the same as the chance of getting different values when picking two values out of a hat (with replacement). The general case is even more difficult to interpret, as it involves a zero cumulative directional difference between agreement and getting the agreed value and some other value when picking two values out of a hat (with replacement).

6.2 Two non-nominal observations per unit

Let us now consider the case where every unit receives exactly two observations (whether by the same two observers for all units or not), but where disagreements are not all of equal severity.[6] We may first note that all the statistics discussed so far, except AC_1, have been formulated above in a way that weights each cell in a contingency or coincidence array using the δ_{ij} function, which (as defined above) assigns weight 0 to all agreements and weight 1 to all disagreements, as is proper

[6] For a simple example, if the observations represent distances, then a disagreement by a mile is usually much more severe than disagreement by an inch.

for nominal values. Let us restate these formulations:[7]

$$
\begin{aligned}
& & D_o &= \frac{1}{n_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} n_{ij} \delta_{ij}, \\
\alpha &= 1 - \frac{D_o}{D_e^\alpha}, & D_e^\alpha &= \frac{1}{o_{..}(o_{..} - 1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}, \\
\pi &= 1 - \frac{D_o}{D_e^\pi}, & D_e^\pi &= \frac{1}{{o_{..}}^2} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}, \\
\kappa &= 1 - \frac{D_o}{D_e^\kappa}, & D_e^\kappa &= \sum_{i=1}^{m} \sum_{j=1}^{m} p_i q_j \delta_{ij}.
\end{aligned}
$$

Since $P_e^{AC_1} = D_e^\pi / (m-1)$, one might define a δ-based AC_1 by importing the δ-based P_e^π; however, Gwet (2012, p. 89) gives a different definition. I will not discuss AC_1 further in this subsection.

If we allow δ to vary from the previous definition, in any case requiring its values to be nonnegative, κ becomes Cohen's (1968) *weighted kappa*, which is, in formulas, often written as κ_w. Cohen himself did not give much guidance on how to choose an appropriate definition of δ, remarking only that it "can be assigned by means of any judgment procedure set up to yield a ratio scale, [often] a consensus of a committee of substantive experts or even, conceivably, the investigator's own judgment" (p. 215). A number of particular definitions for δ (often called v in the weighted-kappa literature) have been suggested in the literature; for example, the textbook of Fleiss, Levin, et al. (2003, p. 609) mentions essentially the following alternative definitions: $\delta_{ij} = |i - j|$, often called the *linear weights*, and $\delta_{ij} = (i - j)^2$, often called the *quadratic weights*. Fleiss and Cohen (1973) have proven that the weighted kappa, using the quadratic weights, is equivalent to the intraclass correlation coefficient that includes a component for "systematic variability between raters" (p. 617); Krippendorff (1970) apparently also proved this result without realizing its relevance to the weighted kappa.

Krippendorff (2011, 2013) discusses a number of definitions of δ (which he calls "difference" or "metric") for use in his alpha. He requires for each such definition that $\delta_{ii} = 0$ for all $v_i \in V$ and that $\delta_{ij} = \delta_{ji}$ for all $v_i, v_j \in V$. In the case of a finite set of values $V = \{v_1, \ldots, v_m\}$ having a relevant total order (which, for notational convenience, I assume corresponds to the natural ordering of the subscripts), Krippendorff (2013, p. 288) defines what he calls the *ordinal metric*, for all $k, \ell \in \{1, \ldots, m\}$, by

$$
\delta_{k\ell} = \begin{cases} \delta_{\ell k}, & \text{if } k > \ell, \\ \left(\sum_{i=k}^{\ell} o_i - \frac{o_k + o_\ell}{2} \right)^2, & \text{otherwise.} \end{cases}
$$

[7] The formula for D_e^κ is stated here the first time, but it is proven as a part of Lemma 37 (p. 251) in the appendix.

Thus, if all the observations in the sample are placed on a single line sorted according to their semantic ordering, the severity of a disagreement is determined by the average number of observations between any two disagreements involving the same pair of values.

When $V \subseteq \mathbb{R}$, such that $v_i = i$ for all $i \in V$, there are several additional ways to define δ for Krippendorff's alpha. One can have $\delta_{k\ell} = (k-\ell)^2$, which Krippendorff (2013, p. 289) calls the *interval metric*, or $\delta_{k\ell} = [(k-\ell)/(k+\ell)]^2$, which Krippendorff (2013, p. 290) calls the *ratio metric* and which scales the severity of disagreement by the magnitude of the two values, so that small differences among small values are comparable to large differences among large values. Krippendorff (2011) also defines two other metrics which I do not discuss here.

I am not aware of any use of a weighted Scott's pi in the literature, even though it is no more difficult to define. Considering the relative similarity of Scott's pi to Krippendorff's alpha, this is not very surprising, as using the former instead of the latter may not add much value. However, Krippendorff (1970) proved that a weighted Scott's pi, using the interval metric (which is essentially the same as the quadratic weights used with weighted kappa), is an intraclass correlation coefficient.

6.3 Arbitrary number of observations per unit

Let us now consider the general case where the number of observations per unit is not restricted to exactly two. Recall that the units are designated $u_1, \ldots, u_n$; designate now, for each u_i, the number of observations the sample contains for it by N_i, such that the observations of u_i in the sample are $c_{i,1}, \ldots, c_{i,N_i}$, and let $N = \max_i N_i$. Without loss of generality, we may assume that $N_i \geq 2$, since any unit that has zero or one observations will not contribute to reliability assessment and can be dropped from the sample.

If we assume that $N_i = N$ for all $i = 1, \ldots, n$, we can (like Mielke et al. 2008) generalize the two-dimensional contingency tables as N-dimensional tables, and the row and column totals ($n_{k\cdot}$ and $n_{\cdot\ell}$, respectively) into totals of $N-1$-dimensional subtables; write n^i_j for the sum of all cells in the subtable obtained by holding the ith dimension at j, so that in the special case of $N = 2$, we have $n_{k\cdot} = n^1_k$ and $n_{\cdot\ell} = n^2_\ell$. Let us continue to write $n_{\cdot\cdot}$ for the sum of all cells in the full N-dimensional table. Now, the proportion of observed disagreement can be defined as

$$D_o = \frac{1}{n_{\cdot\cdot}} \sum_{k_1=1}^{m} \cdots \sum_{k_N=1}^{m} n_{k_1,\ldots,k_N} \delta_{k_1,\ldots,k_N}$$

and the proportion of agreement expected under the assumption that each of the N observers are drawing from their own hats as

$$P^\kappa_e = \frac{1}{n_{\cdot\cdot}{}^N} \sum_{k_1=1}^{m} \cdots \sum_{k_N=1}^{m} \delta_{k_1,\ldots,k_N} \prod_{\ell=1}^{N} n^\ell_{k_\ell}$$

For the unweighted case, define

$$\delta_{k_1,\ldots,k_N} = \begin{cases} 0, & \text{if } k_1 = \cdots = k_N, \\ 1, & \text{otherwise.} \end{cases}$$

Mielke et al. (2008, Eqs. 4–5) generalize existing two-dimensional weights δ_{ij} by defining

$$\delta_{k_1,\ldots,k_N} = \sum_{i=1}^{N} \sum_{j=1}^{N} \delta_{k_i,k_j}.$$

These definitions yield a straightfoward generalization of Cohen's kappa to more than two coders.

Now coincidence tables, and by extension Scott's pi, Krippendorff's alpha, and Gwet's AC_1, are extended to an arbitrary number of observations per unit in a much simpler manner (see Krippendorff 2013, p. 280–287): redefine[8]

$$o_{k\ell} = \sum_{i=1}^{n} \frac{\sum_{a=1}^{N_i} \sum_{b=1}^{N_i} [a \neq b]\,[c_{i,a} = v_k]\,[c_{i,b} = v_\ell]}{N_i - 1}$$

so that $o_{k\ell}$ counts the number of ways the occurrences of the values v_k and v_ℓ in the sample can be paired within units (but not across units), that is, the number of co-incidences of v_k and v_ℓ within units (hence the name *coincidence table*). Note that each observation can be paired with any other observation of the same unit twice; to make each observation count exactly once, the count of pairings is divided by $N_i - 1$. One must further convert

$$r_k = \frac{o_k}{o_{\cdot\cdot}}$$

from a result to a definition.

These (re)definitions result in the generalizations of Gwet's AC_1 and Krippendorff's alpha that the authors themselves have given (Gwet 2008; Krippendorff 2011, 2013), and, in the special case $N_i = N$ for all $i = 1, \ldots, n$, in a generalization of Scott's pi that is known in the literature as Fleiss's (1971) kappa. I am not aware of any use of the full generalization of Scott's pi in the literature.

6.4 Discussion

I have now explored the mathematical properties of four chance-corrected statistics of coder agreement. What remains is to discuss their implications and to determine recommendations of use.

[8] The Iverson (1962, p. 11) bracket notation $[P]$, where P is a truth-valued formula, denotes 1 if P is true and 0 otherwise (see also Knuth 1992).

Since the chance agreement used in Gwet's AC_1 is proportional to the chance disagreement used in Scott's pi, these two statistics must be considered totally incompatible. Since Krippendorff's alpha differs appreciably from Scott's pi only with small samples, the same incompatibility to AC_1 applies to alpha as well; similarly, since Cohen's kappa is identical to Scott's pi when the coders do not disagree on the distribution of values, it too must be considered incompatible with AC_1. Therefore, the same study should never use Gwet's AC_1 together with any of the other statistics considered here.

The greatest difficulty with the three non-AC_1 statistics is what Feinstein and Cicchetti (1990, p. 544) called "the first paradox of κ" (Remark 11, above): all the three statistics will be low even when the coders are extremely accurate, if the distribution of values is severely non-uniform. The same behaviour is visible in the characterization of the three statistics' zero (see Proposition 22): a zero statistic value can result not only from an absence of difference between observed and expected agreement but also from a difference of arbitrary magnitude so long as there is an equal amount of difference in favour of both observed and expected agreement.

Gwet designed AC_1 with the intent of correcting for that behavior. Indeed, when the value distribution is uniform, AC_1 is identical to π (see Proposition 18). The philosophical basis of AC_1 is the dictum of Grove et al. (1981, p. 411) that random assignment of values, assumed in the older statistics' measures of agreement by chance, does not model the actual behavior of coders. This statement is, of course, true, but it is also, in my view, beside the point. The π, κ, and α statistics aim to measure how much the behavior of coders deviate from random assignment, and thus their P_e functions much like a null hypothesis, something that models not what the coders do but what one fervently hopes the coders *do not* do. In contrast, $P_e^{AC_1}$ is a combined estimate of how much the coders have actually guessed and what the effect of those guesses is, thus effectively weighting the chance agreement by the estimated amount of guessing involved. This complexity is reflected in the characterization of when AC_1 is zero given in Propositions 23 and 24, which appears to be hard to interpret.

Further, AC_1 does seem to take a rather simplistic view of what guessing actually entails. The other statistics assume that the observed distribution of values is the distribution used in random selection of values, while AC_1 assumes that guessing involves a uniform distribution at all times. It is not, however, unreasonable to assume that a coder has a prior distribution of values in mind; for example, a coder of the programming language literature is likely to be aware that in general, experimental studies are very rare, and this awareness may affect their propensity to assign a paper the experimental value.

Since AC_1 is identical to π when the value distribution is uniform, there does not seem to be any reason to prefer it in that case. Since AC_1 treats the observed nonuniformity of values as an indication of increased reliability, there seems to be good reason to avoid using AC_1 in the nonuniform case. In my view, therefore, AC_1 should not be used.

The "the first paradox of κ" (Feinstein and Cicchetti 1990, p. 544) discussed

above remains a problem for the three other statistics, but it should be noted that it is a direct result of correcting agreement with an estimate of chance agreement based on the observed distribution of values; despite its name calling out κ specifically, the paradox is shared by all three statistics. It appears, therefore, that the inability to achieve high statistic values under very nonuniform value distributions reflects the difficulty of distinguishing between chance agreement and non-chance agreement under those circumstances; it would seem that it is a feature of the coding situation and not an artefact of the statistics, and is not something that one should count against them.

Of the three remaining statistics discussed here, Cohen's kappa stands apart. It alone assumes that there are exactly two coders, and it alone computes expected agreement by assuming that values are picked from two separate hats with different distributions, each coder using their own hat. If the two hats have the same distributions, κ becomes equal to π, and but if the distributions differ, κ grows beyond π. Difference in distributions reflect systematic disagreement about the boundaries between the values, and Cohen's kappa deliberately discounts this sort of disagreement; but, in my view (also expressed by Krippendorff 2013, p. 303), this sort of disagreement is certainly evidence of unreliability of the coding process, which makes Cohen's kappa inappropriate for evaluating the reliability of coding.

There is not, so far, as clear a case to prefer Krippendorff's alpha or Scott's pi (or its generalization Fleiss' kappa) over the other. With large samples, they are approximately equal. The former is always (disregarding the edge case $\alpha = \pi = 1$) larger than the latter, stemming from their disagreement about whether chance disagreement is better modeled as random selection without or with replacement, respectively, within a unit. Krippendorff (2004, 2013) argues that this difference makes α correct π for small sample sizes; similarly, it can be argued (as I did above) that values within a unit are not independent and thus without replacement is the proper choice. These arguments tip the balance slightly in favor of Krippendorff's alpha.

The choice of α over π becomes more obvious when one leaves the confines of nominal data and the restriction to exactly two observations per unit. Krippendorff has proposed a family of alpha statistics that are theoretically closely related and deal with almost any circumstance that can be encountered in evaluating the reliability of coding: Krippendorff (2013, Chapter 12) discusses α variants dealing with ordinal, interval, and ratio scaled values; Krippendorff (2011) additionally considers circularly and bipolarly scaled values; Krippendorff (1992) also proposed variants allowing for composite values (both sequences and unordered sets). Krippendorff (2013, Chapter 12) has also extended alpha beyond coding, to consider the reliability of unitizing decisions (that is, the selection of many small segments of a large text for coding). No such systematic generalization of Scott's pi exists so far as I am aware (unless one counts Krippendorff's alpha as such a generalization).

Once a statistic has been chosen, it remains to determine a proper interpretation for its values. The values 0 and 1 are generally easily interpreted: the former

means agreement indistinguishable from chance, and the latter means perfect agreement. Occasionally, a statistic may be undefined, which usually indicates a problem in the coding data. But even though a positive value is a minimum requirement for most uses, it is not at all clear what is an adequate level of agreement as indicated by the chosen statistic. A widely cited verbal scale, assigning suggestive words such as "slight" and "substantial" to various ranges of statistic values, was given by Landis and Koch (1977, p. 165), but they readily acknowledge that their scale is "clearly arbitrary" and is not based on any attempt at empirical or theoretical calibration. Krippendorff (2004, 2013) gives, for his alpha, the following guidelines (2004, p. 429)—

> "To assure that the data under consideration are at least similarly interpretable by two or more scholars (as represented by different coders), it is customary to require $\alpha \geq .800$. Where tentative conclusions are still acceptable, $\alpha \geq .667$ is the lowest conceivable limit"

—and indicates that these guidelines are based on experimental experience, although I have not been able to find any detailed published reports of that experience. In the absence of any reliable calibration, I hesitate to recommend acceptability thresholds (beyond zero) myself; however, I consider it important to compute and report the chosen statistic so that the reader can evaluate the situation for themself in light of their own view of the statistic and in light of the methodological guidance that will be available at the time of the reading.

It should be noted that the same statistics are in significant use in widely different application areas with potentially different criteria of suitability. My analysis is focused on their usage in assessing the coding of research articles for the purposes of classifying them, which is closely related to the coding tasks of content analysis (see, e. g., Krippendorff 2013). However, much of the methodological literature, including much of the pioneering research, comes from evaluating agreement about diagnosis in clinical studies, in which context the term "rater" is preferred over "coder". Subtle differences in these usage contexts may create different (often hidden) assumptions which then cause difficult-to-resolve differences of opinion among methodologicians in the different disciplines. One particular point well visible in the literature is whether the identity of the observers ought to matter or not (and hence, whether Cohen's kappa is appropriate or not), which was debated by Krippendorff and Fleiss (1978) in published correspondence; my own analysis supports Krippendorff's position that the identity of observers ought not to matter.

In summary, I consider it a good idea to compute and report Krippendorff's alpha (which I prefer), Scott's pi, or Fleiss' kappa when assessing the reliability of coding in secondary studies. I recommend avoiding Cohen's kappa and Gwet's AC_1 for that purpose. I do not find it appropriate to set in advance a specific, uniformly applicable threshold of acceptability for the value of alpha or pi, except that the value should be positive. Each study, and each reader of a study report, can decide for themself what is an acceptable value for their purposes.

Next, I will discuss the mapping study that I did.

7 THE MAPPING PROCESS

How much empirical research is there that could guide a programming language design process to result in a language as useful to the programmer as possible? To answer this question, I conducted a systematic mapping study, which I reported in my Licentiate Thesis (Kaijanaho 2014). In this chapter, which I largely reused verbatim from that thesis, I explain the actual research design of that mapping study.

The main research question is

RQM: 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 my mapping study 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 mapping study, the following two terminological definitions:

Definition 25. *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 26. *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:

RQM1: How much has the efficacy of particular programming language design decisions been empirically studied?

RQM2: Which programming language design decisions have been studied empirically for efficacy?

RQM3: 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:

RQM4: Which empirical research methods have been used in studying the efficacy of particular programming language design decisions?

RQM5: How common are follow-up or replication studies, either by the original researchers or by others?

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.

7.1 Overview

A high-level view to the mapping process is shown in Figure 2.

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. The details

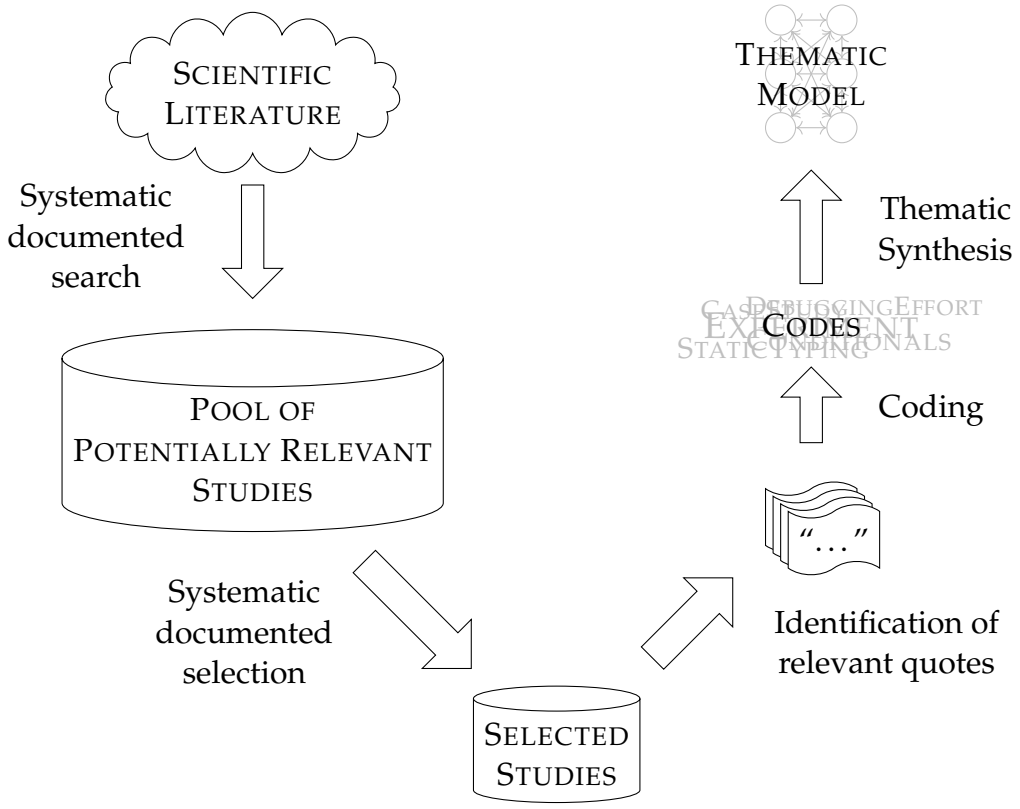

FIGURE 2 A high-level representation of the mapping process. This diagram omits many details.

of the database and the tools I used have been documented in Appendix 1 to my licentiate thesis (Kaijanaho 2014).

I searched for candidate studies by manual and automatic search of various venues and databases and in several iterations, as discussed in Section 7.2. 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 7.3.

After final selection decisions, I conducted a four-stage thematic synthesis process, as discussed in Section 7.4. 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.

7.2 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]

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

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.

7.2.1 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:

- 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)

[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.

- 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)

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.

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.

I considered and rejected several conferences, including the ACM SIGPLAN conference on Programming Language Design and Implementation (PLDI) and the ACM SIGPLAN International Conference on Functional Programming (ICFP), based on my informal experience at the time I designed this study that they are unlikely to contain empirical studies of the kind that would be included in this mapping study. I expected the automatic and snowball searches to correct me in this if I made a mistake (as they indeed did in the case of the IJMMS).

7.2.2 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:

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

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.

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. 149).

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

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

(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:

- 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.

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

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.

7.2.3 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.

7.2.4 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 86). 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 8.3.2.2.

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,

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						

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 the corresponding 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 %

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 %

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 7.4, 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 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.

7.3 Selection

Every publication located during the searches was subjected to a three-phase selection decision procedure, summarized in Figure 3. 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.

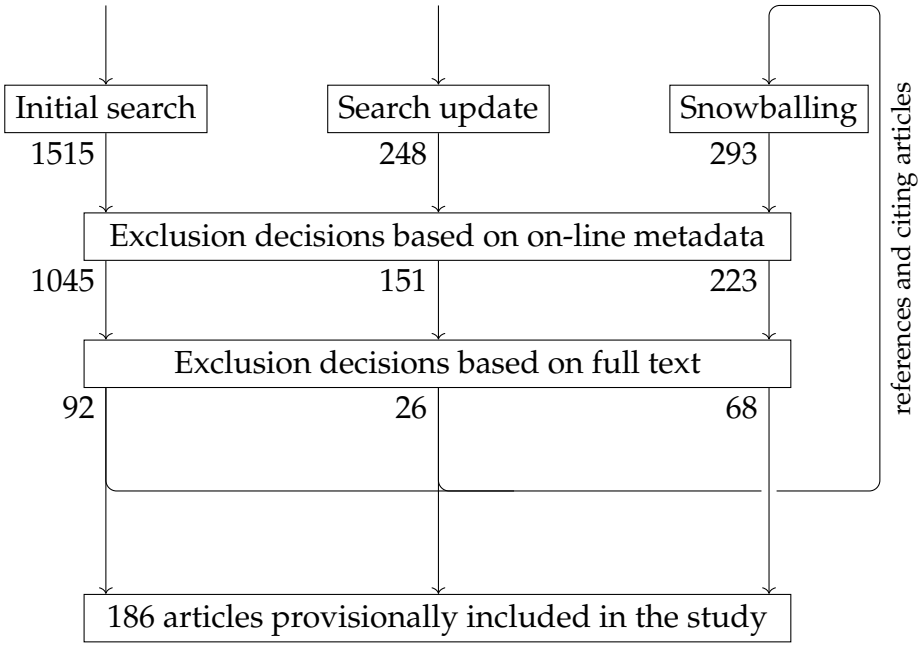

FIGURE 3 Flow diagram of the study selection process. This diagram does not show selection validation and the resulting changes in selection decisions, nor does it show exclusions made *post hoc* during data extraction.

7.3.1 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.)
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?

[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.

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 and Analyses 1 (on page 24), 9 (on page 78), 25 (on page 109), and 26 (on page 109), as well as the following definition:

Definition 27. *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.*

7.3.2 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. In 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.

7.3.3 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 and Koch (1977, p. 165) verbal scale of strength of agreement, 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).

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

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.

As I concluded in Chapter 6, Cohen's kappa should not be used to asssess coder reliability. I have, therefore, reanalyzed this data using Krippendorff's alpha. For the 2011 exercise, $\alpha = 0.48$ (95 % CI 0.16 to 0.74, $n_{..} = 30$), and for the 2013 exercise, my test-retest $\alpha = 0.82$ (95 % CI 0.65 to 0.96, $n_{..} = 200$), and intercoder (disregarding my retest) $\alpha = 0.52$ (95 % CI 0.26 to 0.77, $n_{..} = 87$).

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 to my licentiate thesis (Kaijanaho 2014). The included secondary studies 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.

7.4 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.

7.4.1 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 Kaijanaho 2014, Appendix 6) and intended to synthesize results from the extracted data.

As described in Subsection 7.3.3, 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 8.3.1), 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.

7.4.2 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.

7.4.3 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 to my licentiate thesis (Kaijanaho 2014), and the code assignments themselves are reproduced in its 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.

7.4.4 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 Kaijanaho 2014, 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.

But that brings up the results of the mapping study, which can be found in the next chapter.

8 AN EVIDENCE MAP

Overall,[1] 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 4[2] 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 the Appendices 3 and 4 of my licentiate thesis (Kaijanaho 2014). 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-studies of secondary studies.

---

1 This chapter is largely reused verbatim from my Licentiate Thesis (Kaijanaho 2014).

2 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

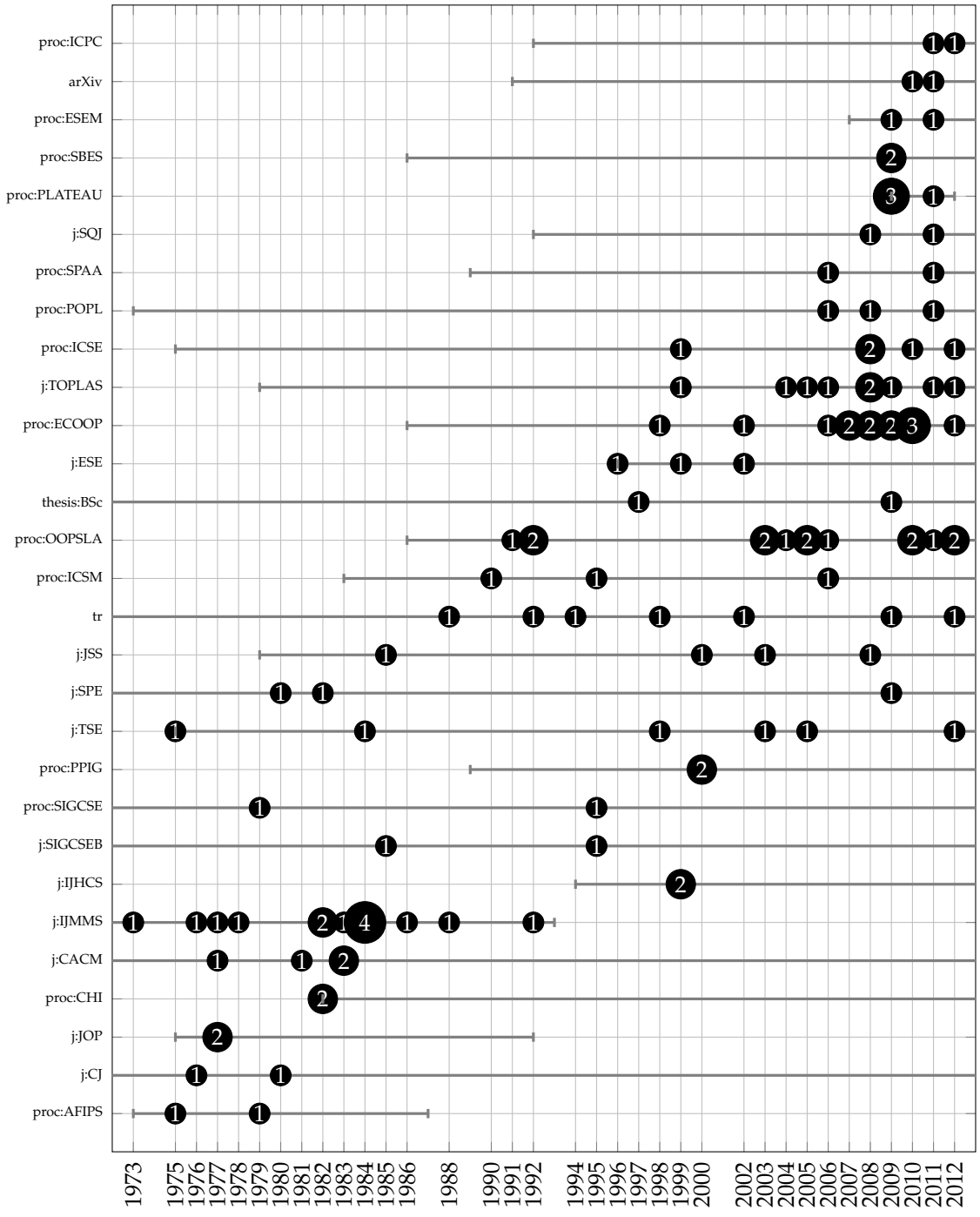

FIGURE 4 Bubble plot of included publications by publication forum and publication year, restricted to forums containing at least two included publications. The years when the forum has been available are indicated.

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	de 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, Zhu, 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

I have used the secondary studies in the validation of the search and selection processes (see Subsections 7.2.4 and 7.3.3). I will only consider primary studies from now on.

8.1 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 to my licentiate thesis (Kaijanaho 2014), and the code assignments are given in its Appendix 3.

8.1.1 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 substudies); 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 BenchmarkPro-

grams, 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.

8.1.2 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 5 shows a version of Figure 4 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.

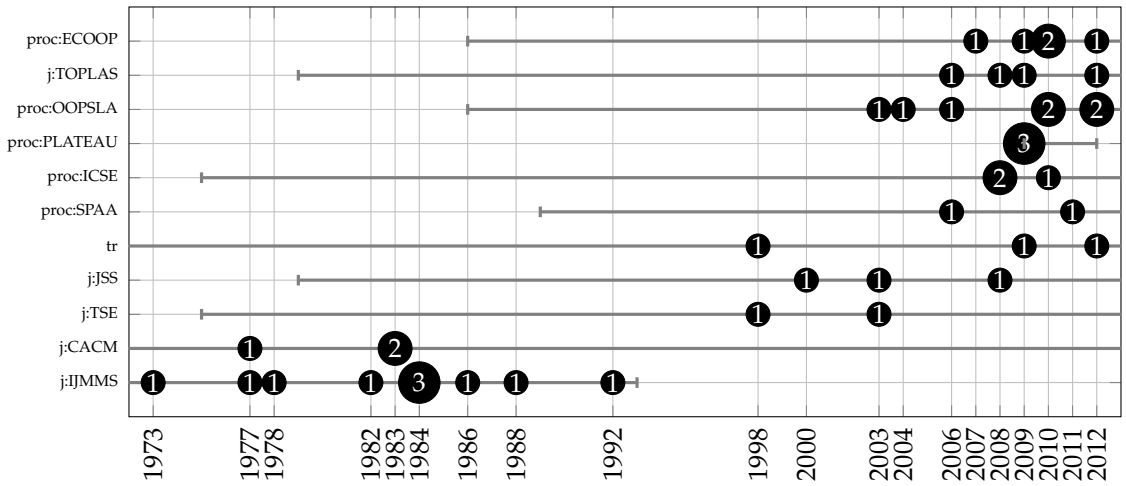

FIGURE 5 Bubble plot of included core publications by publication forum and publication year, restricted to forums containing at least two included core publications. Note that absent years in this plot do not necessarily signify publication-less years. The years when each forum has been available are 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 6 cross-tabulates the facets of efficacy and primary research method used in each sub-study in this outer layer.

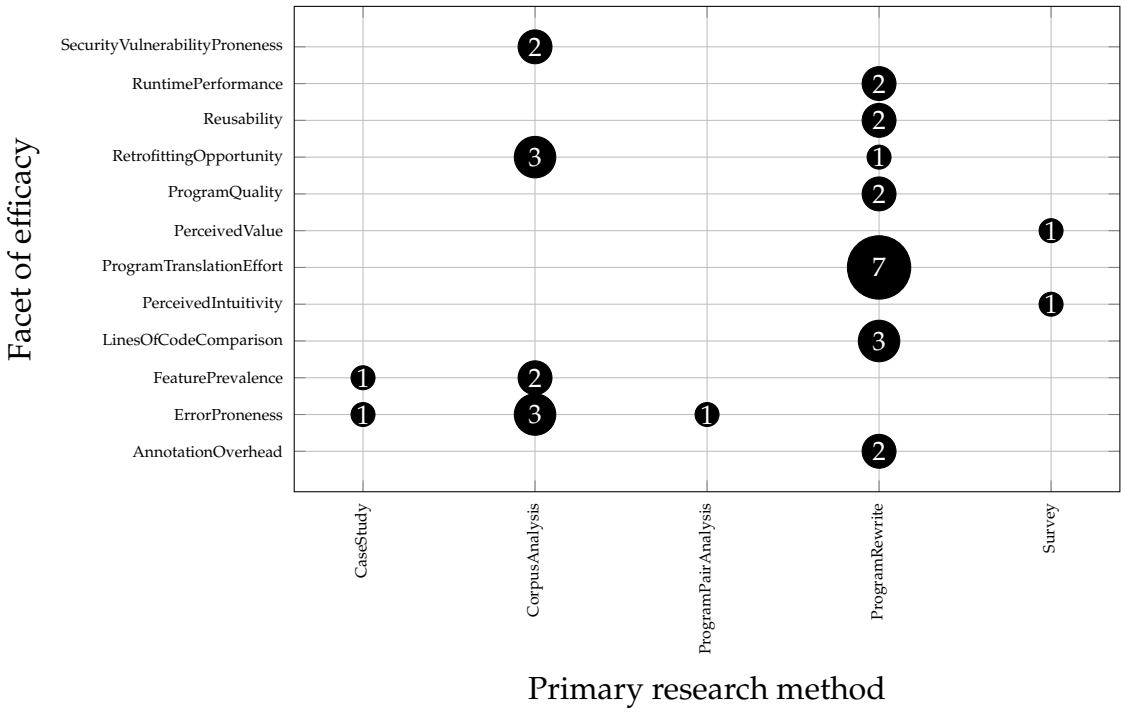

FIGURE 6 Bubble plot of core sub-studies, excluding experiments, categorized by the facets of efficacy used and the primary research methods used.

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, Zhu, 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)

8.1.3 Temporal pattern

Figure 7 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.

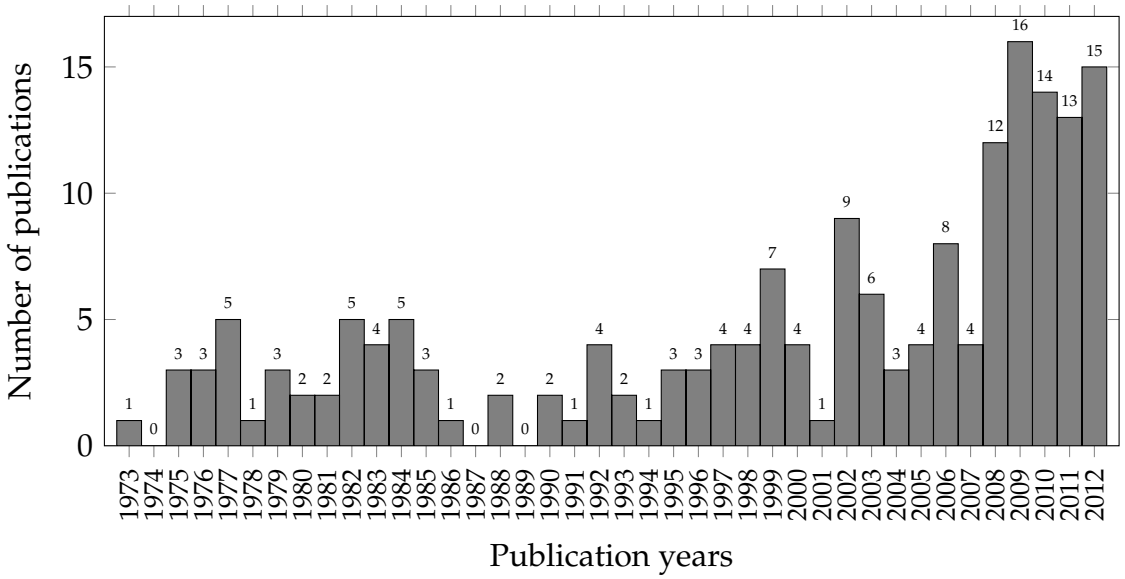

FIGURE 7 The number of included publications per year

Figure 8 summarizes the years of first publication of all primary studies. The same pattern as noted with Figure 7 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 9 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 7 and 8 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 10 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 9, 8 and 7 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.

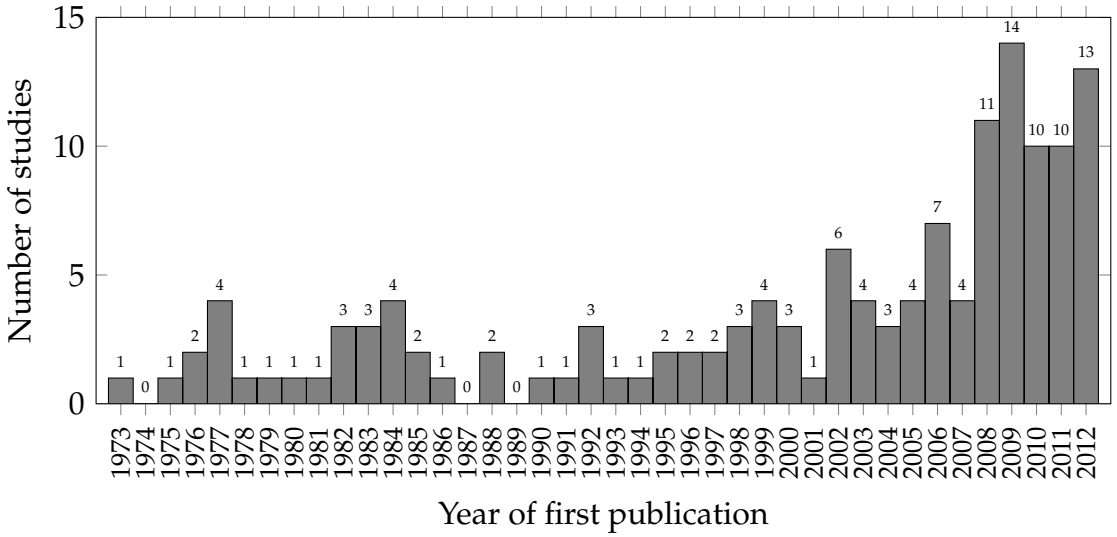

FIGURE 8 The number of included primary studies per publication year

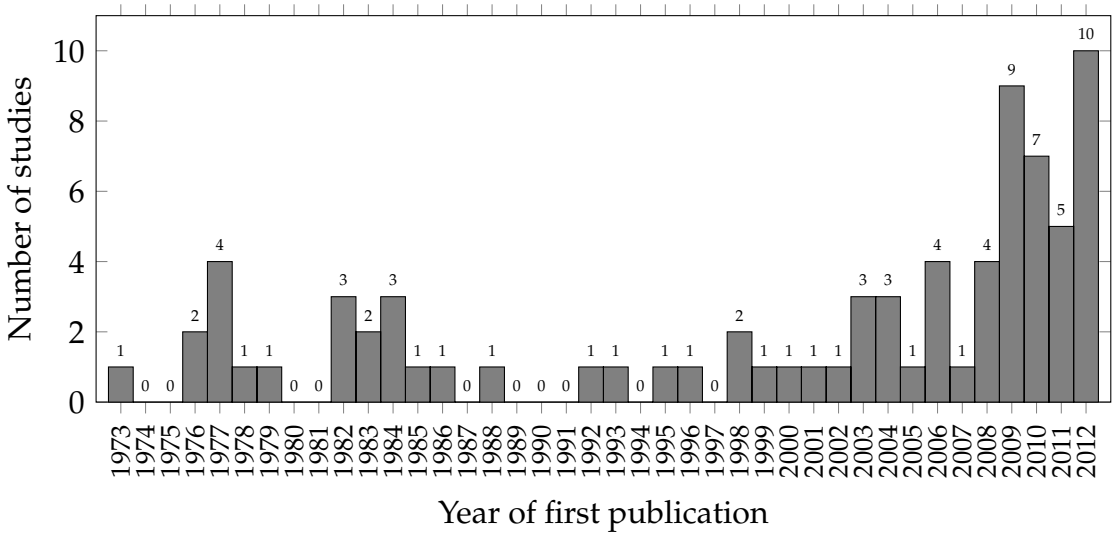

FIGURE 9 The number of included core studies per publication year

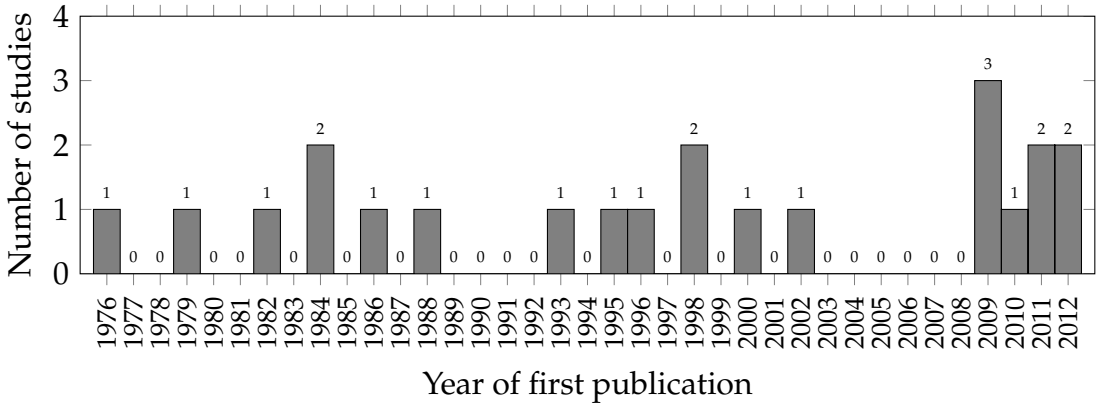

FIGURE 10 The number of randomized controlled experiments in the core per publication year

8.2 Answers to research questions

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

RQM1 *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.

RQM2 *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 11, 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.

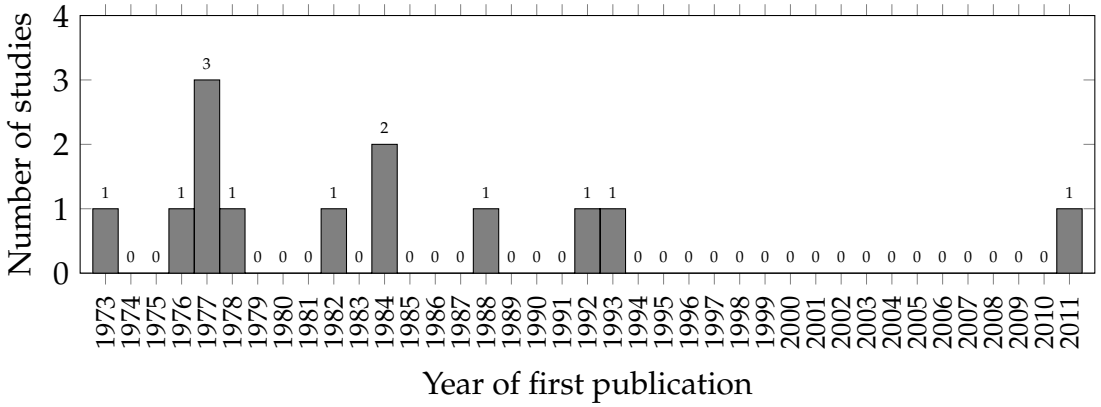

FIGURE 11 The number of core studies of conditionals per year

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 11 shows the distribution of the studies over the years.

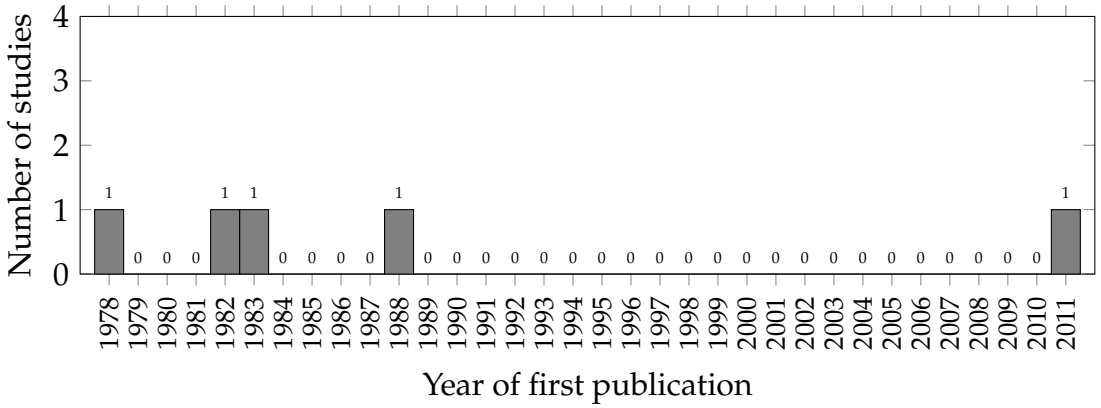

FIGURE 12 The number of core studies of loops per year

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

RQM3 *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 6, 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.

RQM4 *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[3] definition (Yin 2009; Runeson, Höst, 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.

RQM5 *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.

---

[3] 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		

8.3 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 noticeable 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.[4] The 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 (1962/1996) sense, has emerged to capture the imagination of a generation of researchers; again, disregarding the recent years.

[4] In discussions with currently active researchers in this area, particularly during Dagstuhl seminar 15222, another explanation surfaced: the resistance of referees may be making it difficult to get empirical research in this area published (cf. Tichy 2000).

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.

8.3.1 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, following the example of medicine and other fields, 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 adopted this practice in the abstract of my Licentiate Thesis (Kaijanaho 2014).

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 7.4.1, 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.

8.3.2 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.

8.3.2.1 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.

8.3.2.2 Literature search and selection

The quasi-sensitivity of automatic search was, as reported in Subsection 7.2.4, 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 7.2.4) 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 7.3.3 show Krippendorff αs arouns 0.5 for intercoder agreement, which is fairly low but accords with the experience of Stefik, Hanenberg, et al. (2014) and is well above the hard limit of zero.

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.

8.3.2.3 Thematic synthesis

The validation exercise of coding, described in Subsection 7.4.3, 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.

Next, I will go and develop the idea of evidence-based programming language design. The results of this mapping study will find some use there.

9 EVIDENCE-BASED ____

My goal in this chapter is to present and argue for a specific analysis of the concept *evidence-based* _____ (hereinafter abbreviated as EB*x*), in preparation for transporting it to the programming language context in the next chapter. I will start by making the preliminary point that EB*x* has been analyzed before, and therefore any analysis I make should not start *de novo* but from examining these existing analyses, both explicit and emergent.

9.1 Evidence-based medicine

The oldest explicit analysis of this concept, or rather its explication, was published in 1992 (Evidence-Based Medicine Working Group 1992) in the context of on-the-job training of medical doctors (called *residence*). Gordon Guyatt, the chair of the Working Group, reports in an interview by Barends and Briner (2014) that the concept was invented by David Sackett for use in a medical school course about a decade earlier; the name was Guyatt's. The Working Group contrasted "Evidence-Based Medicine", their new model, to the "way of the past": dealing with a patient that has a specific problem the resident is unsure of how to handle, the resident was supposed to consult more senior colleagues and the supervising (or *attending*) doctor, who then gave experience-based advice, but now they are expected to engage in (p. 2421, a bibliography reference omitted)

> "precisely defining a patient problem, and what information is required to resolve the problem; conducting an efficient search of the literature; selecting the best of the relevant studies and applying rules of evidence to determine their validity; being able to present to colleagues in a succinct fashion the content of the article and its strengths and weaknesses; and extracting the clinical message and applying it to the patient problem."

The Working Group claimed this to be a Kuhnian (1962/1996) paradigm shift.

This original explication is echoed, with minor changes and some improvements, in many later articles and books expounding evidence-based medicine. For example, Rosenberg and Donald (1995) offer the following analysis (p. 1122):

> "Evidence based medicine is the process of systematically finding, appraising, and using contemporaneous research findings as the basis for clinical decisions. [...There are f]our steps in evidence based medicine[:]
>
> "– Formulate a clear clinical question from a patient's problem
> "– Search the literature for relevant clinical articles
> "– Evaluate (critically appraise) the evidence for its validity and usefulness
> "– Implement useful findings in clinical practice"

A consensus statement arising out of an international conference, called the "Sicily statement", while recommending the renaming of EBM as Evidence-Based Practice (EBP), offered the following analysis (Dawes et al. 2005, bibliography references omitted):

> "The five steps of EBP were first described in 1992 [...]
>
> "1. Translation of uncertainty to an answerable question
> "2. Systematic retrieval of best evidence available
> "3. Critical appraisal of evidence for validity, clinical relevance, and applicability
> "4. Application of results in practice
> "5. Evaluation of performance
>
> "This five-step model forms the basis for both clinical practice and teaching EBP"

It is unclear where the fifth step originates, as it was not clearly specified by the Working Group (althoug it would be present in an educational context in any case), nor was it included by Rosenberg and Donald (1995).

An influential textbook (Straus et al. 2011) offers two analyses. The first is from a section called "What is EBM?" (p. 1):

> "Evidence-based medicine (EBM) requires the integration of the best research evidence with our clinical expertise and our patient's unique values and circumstances."

The second begins "[t]he complete practice of EBM comprises five steps" (p. 3) and then repeats the five steps of the Sicily statement with a somewhat different set of word choices. An interesting point Straus et al. make is that some of the steps may be omitted while still practicing EBM: the full procedure is reserved for the most common problems, for occasional problems one omits step 3 (because one restricts the search to pre-appraised evidence), and for very rare problems (as well as during one's initial training) one omits at least steps 2 and 3; they call these stages "doing", "using", and "replicating" modes of practice.

These sources appear to document a consistent tradition within medicine, starting from the Evidence-Based Medicine Working Group (1992) up to the current times.[1] I will infer from them the following analysis of Evidence-Based Medicine:

Analysis 28. Evidence-based medicine *is*

1. *a decision procedure to resolve uncertainty*

[1] I should note here that Jeremy Howick's (2011) book *The Philosophy of Evidence-Based Medicine* deals only with the question of the evidence hierarchies used in EBM, which I will examine below. Contrary to what one might expect based on the title, it does not have anything to say about the analysis of EBM's definition in the sense I am discussing it here.

2. *regarding a particular patient's medical issue*
3. *applied by an individual medical practitioner*
4. *consisting of five steps:*
   (a) *formulating a question*
   (b) *locating evidence*
   (c) *appraising the evidence*
   (d) *applying the evidence*
   (e) *evaluating one's own performance*

This analysis deliberately leaves the further analysis of "evidence" for later (Section 9.3). It also deliberately ignores the distinction between "doing", "using", and "replicating" suggested by Straus et al. (2011), as it is orthogonal to this analysis.

This analysis is, of course, defeasible, as the argument supporting it is inductive. Potential ways of defeat include showing that this analysis is, in fact, incompatible with my stated sources, showing that there are other sources within the same tradition that contradict this analysis, showing that there is another tradition using the name "evidence-based medicine" which is incompatible with this analysis, and showing empirically that most practitioners disagree with this analysis. I have attempted to locate sources[2] to defeat this analysis in any of these ways, finding (with one notable exception discussed below) none. Regarding the empirical question of practitioner agreement, I have not been able to find any empirical studies investigating that issue.

One might claim that the proper analysis of EBM is not the five-step model. Cohen, Stavri, et al. (2004) take a descriptive statement in an editorial as their definition (Davidoff et al. 1995, p. 1085):

> "In essence, evidence based medicine is rooted in five linked ideas: firstly, clinical decisions should be based on the best available scientific evidence; secondly, the clinical problem-rather than habits or protocols-should determine the type of evidence to be sought; thirdly, identifying the best evidence means using epidemiological and biostatistical ways of thinking; fourthly, conclusions derived from identifying and critically appraising evidence are useful only if put into action in managing patients or making health care decisions; and, finally, performance should be constantly evaluated."

But this is not an analysis, nor a definition; it is merely a statement of requirements. As another example, in a philosophical analysis of EBM, Sehon and Stanley (2003) "take the term 'evidence-based medicine' to refer essentially to the practice of taking RCTs as the strongly preferred form of medical evidence"; but their explicit aim there was to isolate what distinguishes EBM from other forms of practicing medicine, which necessarily deemphasizes all commonalities, however essential to EBM *an sich*.

There is one arguable defeat for this analysis, however. As Bell (2012) and Greenhalgh et al. (2014)—among others—point out, the evidence-based medicine

[2] Among other means, I have searched on Google Scholar for "evidence-based medicine is", as well as used Google Scholar to locate articles citing the sources I have cited; but I have not conducted a systematic secondary study on the matter.

movement has produced such activities as the Cochrane Collaboration[3] and the Finnish Käypä hoito[4] that produce masses of preappraised and predigested guidelines for clinical practice. These activities are not related to any particular patients nor are they applied by individual practitioners, and arguably therefore, contradict the above analysis. However, this is a defeat of the analysis only if these activities actually are what EBM is about; but like Greenhalgh et al. (2014), I regard them as auxiliary support structures, useful for EBM but not its core, and therefore, I do not view this as a defeat of the analysis.

9.2 EB*x* in other disciplines

Outside the health-care disciplines, the EB*x* concept has been explicated several times in different contexts, typically explicitly basing it on the EBM model. For example, a number of management scholars are developing what they call Evidence-Based Manangement or EBMgt, explicitly modeling it after EBM; Briner et al. (2009, p. 19) analyze it as follows:

> "Evidence-based management is about making decisions through the conscientious, explicit, and judicious use of four sources of information: practitioner expertise and judgment, evidence from the local context, a critical evaluation of the best available research evidence, and the perspectives of those people who might be affected by the decision."

They also seek to rebut misunderstandings about the concept by giving "four key points":

> "EBMgt is something done by practitioners, not scholars. [...] EBMgt is a family of practices, not a single rigid formulaic method of making organizational decisions. [...] Scholars, educators, and consultants can all play a part in building the essential supports for the practice of EBMgt. [...] Systematic reviews (SRs) are a cornerstone of EBMgt practice and its infrastructure"

They do not advocate the five-step model of EBM, rather they put forth their own: identification of problem, the gathering and examination of internal evidence, identification and appraisal of external (research) evidence, the solicitation of stakeholder views, followed finally by the decision making.

Similarly, Sherman (1998) describes Evidence-Based Policing (p. 3):

> "Evidence-based policing is the use of the best available research on the outcomes of police work to implement guidelines and evaluate agencies, units, and officers."

He does not describe any variant of the five-step model, but does outline an agency-level process (p. 4–5):

> "Evidence-based policing is about two very different kinds of research: basic research on what works best when implemented properly under controlled conditions, and ongoing outcomes research about the results each unit is actually achieving by applying (or ignoring) basic research in practice. This combination creates a feedback loop [...]

[3] http://www.cochrane.org/

[4] http://www.kaypahoito.fi/web/english/home

> that begins with either published or in-house studies suggesting how policing might obtain the best effects. The review of this evidence can lead to guidelines [which] would specify measurable 'outputs,' or practices that police are asked to follow. [...] The observation that some units are getting better results than others can be used to further identify factors associated with success, which can then be fed back as new in-house research to refine the guidelines"

Pullin and Knight (2001) propose evidence-based conservation (of nature). Their five-step approach is, however, radically different from those I have discussed above (p. 53):

> "The basic steps are as follows: (1) formulate a policy that action should be evidence-based; (2) develop and promote a method of systematic review with provision of funding; (3) identify priority areas for systematic review; (4) identify gaps in knowledge through the review process and prioritize these for research funding; (5) develop mechanisms to promote and maintain the concept of evidence-based practice among practitioners."

They reinforce this departure from EBM in their later works; for example, in Pullin and Knight (2003) they define the function of evidence-based conservation to be (p. 84)

> "(a) the production of systematic reviews of the primary literature, which evaluate the evidence (including its quality) for the effectiveness of alternative actions in achieving stated objectives and then (b) making this information available to decision-makers through active dissemination."

On the same page, they define a five-step process for the decision-maker, the second step of which being "apprais[ing] the evidence provided for them (instead of searching for it themselves)".

In the field of public policy, the term "evidence-based policy" has seen use. Wyatt (2002) traced this term's inception within the United Kingdom government to a political will to reform government coinciding with the rise of evidence-based medicine. Closer to the present, Nutley et al. (2010) describe two variants of it:

> "[1.] a movement that promotes the use of systematic reviews of research studies aimed at assessing the effectiveness of health and social policy interventions and the translation of these into evidence-based intervention programmes, tools and guidelines [...2.] an approach that helps people to make well-informed decisions about policies, programmes, projects and practices by putting the best available evidence at the heart of policy development and implementation"

Kitchenham, Dybå, et al. (2004) suggest the transportation of EBM into sotfware engineering, creating Evidence-Based Software Engineering (EBSE). They define the goal of EBSE as being (p. 273)

> "to provide the means by which current best evidence from research can be integrated with practical experience and human values in the decision making process regarding the development and maintenance of software."

They also explicitly port the five-step model into EBSE. They further note that in addition to an individual practitioner (that is, a software developer), the point of view of managers is of great importance to EBSE.

However, recent research publications that explicitly mention EBSE use it in very diffent senses. For example, Fitzgerald et al. (2014) report a case study in a company using in-house development data to guide management decisions, calling this "evidence-based software project management" and clearly positioning it within the EBSE umbrella; Santos, de Magalhães, et al. (2014) characterize EBSE as a "research approach" proposed by Kitchenham, Dybå, et al. (2004), with systematic reviews being "the preferred method" for "find[ing] and aggregat[ing] evidence from scientific research" (p. 1); Salleh and Nordin (2014) associate EBSE as being influenced by systematic reviews in medicine and consisting of systematic reviewing; and Babar et al. (2014) consider EBSE to be a toolset used in systematic reviewing.

These examples demonstrate a lack of unified view across disciplines of what EBx is. While all of them can be (usually explicitly) traced to an attempt to transport EBM to a different discipline, they have reached quite different explications of their variant of EBx. From them, a concept speciation may be inferred:

Analysis 29. *EBx_1 is*

1. *a decision procedure to resolve uncertainty*
2. *regarding a particular practical problem in the field x*
3. *applied by an individual practitioner of x*
4. *consisting of five steps:*
   (a) *formulating a question*
   (b) *locating evidence*
   (c) *appraising the evidence*
   (d) *applying the evidence*
   (e) *evaluating one's own performance*

This is the EBM and EBSE model, as specified in their foundational sources.

Analysis 30. *EBx_2 is*

1. *a decision procedure to resolve uncertainty*
2. *regarding a particular practical problem in the field x within an organization*
3. *applied by the organization in question*
4. *consisting of five steps:*
   (a) *formulating a question*
   (b) *locating evidence internal to the organization*
   (c) *locating evidence external to the organization*
   (d) *applying the evidence to solve the problem*
   (e) *generating new internal evidence to evaluate the solution*

This is the EBMgt and evidence-based policing model.

Analysis 31. *EBx_3 is*

1. *a systematic effort*

2. *to collect evidence and*
3. *to preappraise that evidence*
4. *for the use of practitioners in x*

This is the approach advocated for evidence-based conservation and one of the senses reported above for evidence-based policy. It also can be seen as an alternative analysis of EBM (which I rejected above), and appears to be an analysis of EBSE some researchers have silently adopted.

These analyses are defeasible, as they follow from an inductive argument. However, as I deliberately used concept speciation and abstraction to limit their applicability and specificity, they are rather weak analyses; I do not see any way for defeat to occur, absent my own error, but I do not see them as particularly useful either. The more important conclusion I infer from the foregoing is that a concept drift has occurred: EBx does not seem to have a coherent definition or analysis that is shared by the community of people adhering to EBx in all the disciplines. Thus, there is no concept of EBx that has given the privileged position of being correct.

I do, however, argue that EBx_1 is the most appropriate generalization of EBM. This conclusion rests first on my earlier asserted analysis of EBM, which I hold to be correct, and second on the fact that EBx_1 merely abstracts away the name of the discipline while all the other variants are based on larger deviations from it (arguably, on misunderstandings of EBM).

In conclusion, since EBx_1 is the best generalization of EBM, it shall be used in this dissertation as the analysis of EBx.

9.3 Based on what evidence?

Earlier, I deferred the analysis of "evidence" as it is used in both EBM and EBx for later; that is the task of this section. I will start with examining what the sources I used to infer my EBM analysis say on this topic.

The Evidence-Based Medicine Working Group (1992) does not explicitly discuss what it means by evidence, but the following occurrences of the word make clear its intent:

> "evidence from clinical research [...] selecting the best of the relevant studies and applying rules of evidence to determine their validity [...] Acting as a role model involves specifying the strength of evidence that supports clinical decisions. In one case, the teacher can point to a number of large randomized trials, rigorously reviewed and included in a meta-analysis, which allows one to say how many patients one must treat to prevent a death. In other cases, the best evidence may come from accepted practice or one's clinical experience and instincts. [...] Careful history taking and physical examination provide much, and often the best, evidence for diagnosis and direct treatment decisions."

The word "evidence" is clearly meant broadly, to include at least published research, current best practice, personal experience, personal instincts and the examination of the individual patient.

Rosenberg and Donald (1995) do not discuss what they mean by "evidence" either, but a similar examination of usage contexts demonstrates clearly that they mean to limit it to published research only. The Sicily statement (Dawes et al. 2005) also leaves evidence implicit but appears to intend it to be limited to published research. Straus et al. (2011) sidestep the problem by qualification: they talk of "research evidence", which they define as "clinically relevant research" (p. 1); in the rest of the textbook, they do use the word "evidence" without qualification, but the context makes clear that this is not meant to be a significant widening of the intended concept; thus, for Straus et al., "evidence" appears to be limited to published research.

As I have inferred Analysis 28 of EBM from these sources, it is not amiss to infer the meaning of "evidence" as a technical term of the analysis from these sources as well. There is an apparent problem: the sources are not unanimous except as to the inclusion of published clinical research. A closer inspection of the apparently dissenting source (Evidence-Based Medicine Working Group 1992) shows, however, that where it discusses the five-step model (which it calls "the critical appraisal exercise" on p. 2421, emphasis deleted) it agrees with the others—the "evidence" searched for and appraised is "the best of the relevant studies" (p. 2421)—and its other uses of "evidence" are not in the context of the five-step model. Therefore:

Analysis 32. *In the context of EBM,* evidence *means relevant published studies.*

As discussed above, evidence-based management and evidence-based policing distinguish between two kinds of evidence: published research and in-house generated data. Briner et al. (2009) refer to them as *external evidence* and *internal evidence*, respectively. A similar distinction might be made within the context of EBM also, in which case internal evidence would correspond to patient-specific information such as "[c]areful history taking and physical examination" (Evidence-Based Medicine Working Group 1992, p. 2423), and thus this distinction may be possible to be extended to EB*x* in general:

Analysis 33. *Within the context of EBx,* external evidence *means relevant published studies, and* internal evidence *knowlede about the specific context in which the EBx decision procedure is being applied.*

Since the EB*x* analysis is derived from EBM, its unqualified use of the word "evidence" should be understood in the same way:

Analysis 34. *Within the EBx analysis,* evidence *means relevant published studies.*

In other words, evidence means external evidence. Note that this conception of evidence is different from that I adopted in Section 4.7; it does, however, approximately accord with the notion of scientific evidence in Definition 9.

9.4 Hierarchies of evidence

The EBM position on evidence can be strengthened further. Howick (2011) argues, and I agree, that the EBM position has been from the start (and remained to the time when he was writing) that

> "(systematic reviews of) comparative clinical studies, preferably randomized trials, provide better evidence than mechanistic reasoning and expert judgment[.]"

This position leads to the notion of a *hierarchy of evidence* in which studies are ranked primarily by their research design, and any study at a higher rank is taken to be more reliable than another study at a lower rank. Many such hierarchies have been proposed; Blunt (2014) lists 88 publications promulgating an evidence hierarchy within medicine, each somewhat different from others. Largely they appear to be formalized for the use in systematic secondary studies and clinical guidelines.

Steps 2 and 3 of the five-step model are both heavily dependent on some sort of evidence hierarchy, much more so than any secondary study or guideline would be. In EBM, the goal is for an individual physician to locate and appraise evidence related to a particular patient or class of patients, and thus a major requirement for both steps is to filter out as many studies as possible without seriously degrading the quality of the answer obtainable from the remaining studies. Any method proposed for Step 2 needs to be reducible to a Boolean search expression or to a list of publication forums deemed acceptable, or a combination of them; and any method proposed for Step 3 must be compatible with the method chosen for Step 2. It is difficult to see how any method satisfying these requirements could be anything but a hierarchy of evidence. Thus, the design of a such a hierarchy is of paramount importance to EBx to the extent that if it is deemed impossible, EBx becomes likewise impossible.

The traditional evidence hierarchy—where a single randomized controlled trial (RCT), no matter how badly conducted, trumps all other kinds of primary studies, no matter how well conducted they are and no matter how dramatic a result they show, as well as reasoning from theoretical knowledge—has been the target of criticism (see, e. g., Cohen, Stavri, et al. 2004; Borgerson 2008, 2009; Goldenberg 2009). Such a hierarchy can, in my view, only be justified by arguing that there is some probability close to 1 (or, alternatively, a deductive necessity) that a single study on a higher level of the hierarchy will be at least as close to the truth as the aggregate result of any set of studies on a lower level, based on some suitable measure of closeness to truth; if such an argument were to be made, it would justify ignoring all studies on all lower levels if at least one study is found at a higher level.

I believe that the traditional hierarchy cannot be so justified. First of all, recall that the historical purpose of randomization in an empirical experiment is to help conceal the allocation and ordering of interventions from the participants and the experimenters (Hacking 1988; Hart 1999; Chalmers 1999), not any mystical belief in randomization eliminating all confounders (as claimed by,

e. g., Fisher 1937; Colquhoun 2011). A badly designed and conducted randomized study will surely be much further from the truth than a well designed and conducted nonrandomized study under any reasonable measure of closeness to truth. For example, if what one measures is irrelevant for the research question, no amount of randomization will save the study.

Further, the traditional hierarchy places theoretical reasoning below randomized trials: there are excellent theoretical reasons to discount the possibility of precognition, but a randomized trial by Bem (2011) reported a statistically significant precognitive effect; similarly, there are excellent theoretical reasons, backed by the results of prior research (see, e. g. House of Commons Science and Technology Committee 2010), to think that homeopathy has no real effect, yet a randomized trial by Saha et al. (2013) reported a statistically and clinically significant reduction in blood pressure in hypertensive patients treated with homeopathy.[5] These extreme examples show, convincingly in my view, that a hierarchy where a distinction is made between RCTs and non-RCTs cannot be reliable.

The Straus et al. (2011) textbook explicitly recommends a search technique involving an abstraction hierarchy suggested by Haynes (2001, 2006) and DiCenso et al. (2009) called 6S: primary *studies* can be abstracted individually by *synopses* or collectively by systematic reviews (*syntheses*), and the latter can be abstracted individually by synopses or collectively by *summaries* such as guidelines and textbooks; all of these can be integrated into a computerized *system*. The 6S hierarchy suggests starting a search in a suitable system, and if that does not answer, then descending to summaries, and so on until, if all else fails, one reaches the bottom of the hierarchy and searches for individual primary studies. The 6S hierarchy is easily justifiable as an effort to reduce the time cost of a single search, though—as Haynes (2006) notes—developing and maintaining the abstractions do carry a substantial cost. Its validity depends—as Haynes (2006) again notes—on that the abstraction methods used are accurate and bias-free; it cannot be evaluated in the abstract. However, it seems plausible that methods can be designed for accurate abstraction of research.

Still, the question remains whether a useful hierarchy of evidence can be devised which, with a reasonably high probability, will filter out studies that mislead and studies that, given other studies present and not filtered out, will not materially contribute to increasing the accuracy of the result. However, it is likely that any such hierarchy must be tailored to take into account the unique features of the discipline in which it is to be applied. I have no answer to offer.

---

[5] I have not evaluated these two studies for their methodological rigor and I thus make no claim as to whether they are "good" or "bad" trials in the sense I used those adjectives above. Due to the statistical nature of these trials, there is no logical inconsistency between the proposition (which I accept) that no real effect exists in either study and the proposition (on which I am undecided) that these two trials are of excellent methodological quality.

10 EVIDENCE-BASED PROGRAMMING LANGUAGE DESIGN

Given my analysis of evidence-based _____, I will in this chapter transport that analysis to the context of programming language design, resulting in *evidence-based programming language design* (*EB-PLD*).

10.1 Existing usage in programming language design

A Google Scholar search on the phrase "evidence-based programming language design" performed on April 12, 2015, finds only one result (Kaijanaho 2014); removing the word "design" yields one additional result (Stefik, Siebert, et al. 2011); removing instead the word "programming" yields a third (Muschevichi et al. 2008); removing both words yields many irrelevant results. The first of these three is my licentiate thesis stating without defending the analysis I will be discussing below.

Stefik, Siebert, et al. (2011) use variants of the phrase "evidence-based programming language" three times, once in the abstract and twice in the introduction. They define it on p. 3 (bibliography references and emphasis omitted):

> "we have used survey methods, usability studies, and field tests to help us design the programming language and make it easier to use. We call such a language an Evidence-Based Programming Language in the sense that we have continuously altered the syntax, semantics, and API design according to both our observational work with novices and results from the literature."

Curiously, the phrase is entirely absent in the journal version (Stefik and Siebert 2013); they seem to have transitioned to using "evidence-oriented language" (Stefik 2014; Stefik, Hanenberg, et al. 2014; Stefik and Ladner 2015). I should also note the phrase "evidence-based language industry" used by some of the same authors (Stefik and Hanenberg 2014).

Muschevichi et al. (2008) use the phrase "Evidence based language design" exactly once, as an unnumbered subsection title within their discussion section.

They remark, citing Hume, that descriptive research cannot answer normative questions but "there seem to be clear advantages to informing the design of future languages with evidence drawn from other than anecdote, personal experience, small-scale observational studies, or personal morality" (p. 579).

Outside of the search, I have also come across a study by van den Bos and van der Storm (2013) in which they gathered a large corpus of image files, then ran a program written in their domain-specific language on that corpus, then modified that program until all problems encountered were solved, and finally reconsidered the design of their language in light of this evidence. This process they dubbed "evidence-based DSL evolution", citing no literature in its support; a look through articles citing it (as shown by Google Scholar on June 19, 2015) show no articles picking that term up.

It is clear that there is no established tradition of evidence-based *x* where *x* is anything related to programming language design. My explication is therefore not constrained by existing usage.

10.2 Overall explication of EB-PLD

Having held on page 158 that EBx_1 will be used as the analysis of EBx in this dissertation, I will take it as the starting point of my explication. Thus, adapting Analyses 29 and 34:

Analysis 35. Evidence-based programming language design *(*EB-PLD*) should be*

1. *a decision procedure to resolve uncertainty*
2. *regarding a particular practical language design problem*
3. *applied by an individual language designer*
4. *consisting of five steps:*
   (a) *formulating a question*
   (b) *locating evidence*
   (c) *appraising the evidence*
   (d) *applying the evidence*
   (e) *evaluating one's own performance*

Here, evidence *means relevant published studies.*

A comment is in order about the phrase "individual language designer". My intent here is not to say that the number of designers matters. The intent is to distinguish between the person or persons actually engaged in a particular language design on the one hand and researchers speculating about language design in the abstract. Thus, EB-PLD is not a process for conducting secondary research but a process for making practical decisions.

An objection might be offered that instead of transporting a concept from other disciplines to programming language design, one should instead give a

de novo analysis within the context of language design, probably starting from the analysis of the words "evidence" and "based". However, it seems best to me to align all uses of EBx to the extent possible and not generate yet another speciation of the concept.

The rest of this chapter continues the explication in the Carnapian tradition. My explicandum is the intuitive idea of applying EBx, as analyzed above, to programming language design. Following Carnap (1962, discussed more fully on p. 47 of this dissertation), my desiderata for the explicatum are

- similarity to the intuitive idea,
- exactness,
- fruitfulness, and
- simplicity.

There is, however, an important difference between Carnap and myself. He was explicating a mathematical theory, while I am explicating a practical method. Thus, exactness will be interpreted as unambiguity in practice (instead of mathematical exactitude) and fruitfulness as having practical (in principle testable) consequences.

In my analysis above, EB-PLD becomes relevant if an individial language designer is faced with a practical language design problem and is uncertain about how to proceed. All of these are important preconditions. In particular, EB-PLD is *not* (directly) relevant to a language design researcher pondering a research problem, or to an organization involved in running a language design project; they may find EBx_2 or EBx_3, which are out of my scope here, more relevant. I will also stress that EB-PLD is not a deterministic design method; rather, it should be called into action only to resolve actual uncertainty.

In his review of this dissertation, Matthias Felleisen points out that a language designer need not wait for the results of their study to be published before making a change suggested by the study to a language. I completely agree. As I argue above, EB-PLD is only to be used when there is actual uncertainty. If an unpublished study well known to a designer resolves their question, then there is no actual uncertainty about it, and the EB-PLD process need not (and should not) be invoked. Similarly, EB-PLD is perfectly capable of coexisting with other approaches to language design; it is not intended to be an exclusive method.

A problem for the further explication of EB-PLD at this time is that there is no consensus on the appropriateness of various research methods in the discipline, and thus it is very likely premature to explicate a detailed evidence hierarchy. However, each language designer is free, of course, to consider their own views on the matter in the absence of such consensus hierarchies.

I will next sketch an explication of the five steps in turn, bearing in mind the aforementioned limitation. This explication must be viewed with caution, as it has not been empirically validated. Following the sketch, I will present two thought experiments designed to illustrate and (to an extent) test the sketch.

10.2.1 Step 1—Formulating a question

I believe it a safe assumption that a language designer has a design sketch at all times. It may be a fairly vague notion of what the designer wants to do, a working, implemented language which the designer is not satisfied with, or something in between. The designer's design problem then defines some set of mutually alternate modifications of that sketch the designer is considering to make, each of which results in a new design sketch; I do not assume that the designer is at this point aware of the precise content of that set of alternatives. Since the designer is, by assumption, starting to use the five-step EB-PLD process, they must have some uncertainty about which of those alternatives to choose.

Later steps will be using the formulated question to guide literature search and selection. Therefore, the purpose of Step 1 must be to define the problem in sufficient detail as to make it possible to locate an answer in the literature, if one exists in it. Without loss of generality one may assume that there is no publication in the literature that directly addresses the designer's problem, since if there were, the language the designer is designer already exists and there is nothing for the designer to do. Thus, the problem must be defined in such a way as to allow the location of publications addressing similar problems whose answers are hopefully transferable to the designer's problem.

The designer must, therefore, identify one or more abstractions of the design problem at hand which encapsulate the designer's problem but is likely to occur in other language designs as well. In the mapping study I called these abstractions *design decisions*. However, for searching for literature to become possible, these design decisions must be identified by names that are commonly used for them.

In order to choose between these alternative design decisions, the designer must identify some *evaluation criteria* for them, for if they do not, then the choice between them is arbitrary and cannot be illuminated by published research. Further, these criteria must be either measurable—whether directly or indirectly—or deducible, for otherwise published research cannot help. At least three categories of evaluation criteria exist:

- *efficacy* or the empirical capability of the design decision to benefit a programmer,
- *efficiency* or the empirical or theoretical capability of the design decision to be implemented with low overhead, and
- *safety* or the theoretical property of the design decision that it not allow undefined behavior.

As shown in Table 17 on page 140, efficacy criteria for which studies exist include error proneness, program comprehensibility, and various programmer effort criteria. Potential efficiency criteria include program runtime. As to safety criteria, type soundness is a typical criterion investigated in the literature. These are examples, and not an exhaustive list; a designer needs to choose criteria that make

sense for their particular design problem. As with design decisions, the identification of the criteria must produce names that are commonly used.

The design decision and evaluation criteria are not enough to form a question; the form of the question is also needed. Two forms appear likely to be of use in EB-PLD:

- What form of the ⟨*design decision*⟩ is best according to the ⟨*evaluation criteria*⟩?
- Which of these alternative ⟨*design decisions*⟩ is best according to the ⟨*the evaluation criteria*⟩?

The first, which I will refer to as a *what question*, is appropriate when the design decision is abstract, and alternatives are sought from the literature; the latter, a *which question*, is appropriate when alternatives have already been identified. I do not claim that this is an exhaustive list. Note that I do not intend these forms to be taken literally; one may rephrase them to suit the problem at hand.

10.2.2 Step 2—Locating evidence

Step 2, that of locating evidence, involves the efficient search of the literature. As it is an empirical optimization problem, its explication is not really suited for philosophical methods, and hence I will limit myself to some tentative remarks only.

In analogue to the 6S model of medicine (Haynes 2001, 2006; DiCenso et al. 2009), it will likely be a good idea to start any search by looking in existing secondary studies that categorize studies by design decision and evaluation criteria, or along other similar dimensions. The mapping study reported in my Licentiate Thesis (Kaijanaho 2014), the results of which are reproduced in Chapter 8 of this dissertation, is one such study. To my knowledge, there is no other similar study in existence.

If no studies are found with the aid of known secondary studies, a fresh search is required. Further, a supplementary search is often warranted to look for studies that are too new to have been taken into account in the secondary studies used, and to cover any known deficiencies of the secondary studies. In both cases, the methods of systematic secondary studies, discussed in Section 5.5, seem like a good starting point, although it is probably not cost-effective to do an exhaustive search in an EB-PLD process.

Following the example of EBM, the designer might assume that randomized controlled trials are the best source of evidence. As discussed in Section 9.4, this sort of an evidence hierarchy may not be well founded. However, it does simplify the search quite a bit.

10.2.3 Step 3—Appraising the evidence

The goal of Step 3 (appraising the evidence) is to find, with a reasonably high probability, a subset of the found studies which suggests an answer to the question at issue that is as close or closer to the truth than that suggested by the full

set of found studies, to a reasonable degree of accuracy. Thus, the goal is to detect studies that are likely to mislead; also, studies that improve the accuracy of the answer to a minor degree only may be dropped.

In EBM, this is conducted typically by selecting a short checklist appropriate to the type of the question and then going through that checklist for each study identified. For example, the checklist given by Straus et al. (2011, Ch. 4) for critically appraising studies of therapy contains three major sections (validity, importance, and applicability) and altogether 13 questions.

In the context of software engineering, a number of quality checklists have been discussed, either for use in systematic reviews (Dybå and Dingsøyr 2008; Kitchenham, Brereton, Budgen, and Li 2009; Kitchenham, Burn, et al. 2009; Ivarsson and Gorschek 2011; Kitchenham, Sjøberg, et al. 2012) or to guide referees before publication (Wieringa et al. 2012).[1] In addition, a case study checklist has been developed for use by case study researchers, reviewers, and readers (Höst and Runeson 2007; Runeson and Höst 2009; Runeson, Höst, et al. 2012). All of them are too abstract, and some of them are too long, to be of use in this context.

In their mapping study, Stefik, Hanenberg, et al. (2014) used the What Works Clearinghouse Standards (What Works Clearinghouse 2014), which are limited to appraising controlled experiments on the measurable effects of interventions but which do have a checklist form; however, Stefik et al. report Cohen's $\kappa = 0.488$ (comparable to the $\kappa = 0.42$ and $\alpha = 0.52$ I have reported in Section 7.3.3), which may suggest that even among programming language researchers the checklist is not very easy to apply and therefore it is not to be expected that a language designer (who may or may not be a researcher by training and experience) will find it easy to apply; however, this may also be an artefact of κ, as discussed in Chapter 6, as almost all studies received the same WWC rating. It is, however, the only study that I am aware of that proposes any appraisal checklist for programming language studies.

I will offer the checklist given in Table 19 tentatively for use with comparative questions. These are questions that ask whether a design decision is better than another, or what would be the best design decision in a particular situation, as evaluated by using specified criteria. The checklist is inspired by the quality appraisal checklist for therapies by Straus et al. (2011, Ch. 4) and, to some extent, the What Works Clearinghouse (2014) standards, but has been heavily modified to account for the differences in the disciplines. I also differ from them by not making randomization a near-mandatory requirement for studies (cf. Section 9.4). In the checklist, the language designer is addressed as "you".

The first two questions examine the research questions of the study in question in detail and verify that the study is truly comparative in a relevant manner. Given that this checklist assumes a comparative question, these questions disregard the possibility that a noncomparative study might be informative. Of course, it is sometimes possible to combine two non-comparative studies to yield a comparative combined result, and the language designer may want to take this

[1] A Google Scholar search for `"quality checklist" "software engineering"` was conducted on April 29, 2015.

TABLE 19 Tentative quality appraisal checklist for comparative questions.

1. Does the study compare language designs differing in a way that is comparable to the difference in potential designs your question is about; and is that difference isolated for study, e. g., by controlling for other differences?
2. Does the study compare the designs using a criterion that is comparable to the evaluation criterion specified in your question?
3. Are the language designs being compared given a materially equal opportunity to succeed at the beginning of the study?
4. Does the study treat the language designs being compared materially identically to the extent allowed by the differences in the designs?
5. Does the data analysis of the study appear to be free from serious problems in general?
6. Is the data regarding the relevant comparison criterion analyzed materially correctly and without material bias in favour or against any of the designs?

possibility into account when applying this checklist item; in such a situation the designer should consider those two studies a single study for the purposes of this checklist.

The first question considers the language design decisions being compared, and the second the criteria of comparison. The word "comparable" used in both questions is deliberately vague. The purpose here is to allow the language designer to reflect on how their own requirements apply to the study at hand, and thus there is no point in specifying an objective set of criteria. The design difference is comparable if the designer feels they can learn something useful from that difference, and similarly the comparison criterion is comparable if the designer feels they can learn from such a comparison. And, consequently, if one or both is not comparable in this sense, there is nothing the designer can learn from the study, and they ought to exclude the study from further consideration.

Note the significant difference between doing this sort of a quality appraisal for one's personal use and doing it for a systematic secondary study. In the latter, the quality appraisal must be objective in the sense that whatever appraisal protocol is used, it should largely yield the same answer to the same input every time, no matter who applies it, so that the answer can be applied again and again by different readers of the secondary study. But in the EB-PLD process, and in EBx generally, the goal is to answer a designer's unique and possibly nonrepeating problem.

The final four questions correspond to Paul Glasziou's race analogy for appraising study validity (Straus et al. 2011, p. 63):

> "First, was there a fair start? [...] Second, was the race fair? [...] Third, was it a fair finish?"

Fair start and fair race get their own detailed question; fair finish is dealt in two questions. In many of these questions, the requirement is tempered by the qualifier "materially", allowing the language designer to judge for themself the se-

riousness of any deviations. If there are material deviations, then the designer considers the study suspect and should exclude it from further consideration.

As to fair start (the third question of the checklist), if equal opportunity is not given in the beginning, then the results of the study are meaningless unless there is some way to compensate for the resulting bias. In an experiment with human participants, random assignment of the participants to designs or sequences of designs is one standard way of providing for equal opportunity, although it is quite possible to deny equal opportunity even with random assignment, for example, by instructing all participants in a biased manner.

As to fair race (the fourth question), if material differences in treatment occurs, then the difference shown by the study may be caused by differences in the language designs' treatment, which makes them suspect. Often it is difficult to spot these issues in a report of a study, but this checklist item exists for those cases where problems are noticeable.

As to a fair finish (the fifth and sixth questions), it involves the manner in which the study results are analyzed and conclusions are drawn. The fifth question is open-ended and largely asks the designer to look for egregious errors, such as wildly inappropriate analysis methods (for example, a philosophical analysis of whether C programmers make more errors than Pascal programmers—a matter better suited for an empirical study) or the gross misuse of statistics, particularly fishing for statistical significance, also called *p*-hacking (see, e. g., Simmons et al. 2011; Motulsky 2015). All these issues are objective ones; there is no subjectivity involved in the fifth question. It could (and should) be further explicated, to identify ways in which analysis can be grossly incorrect, but I leave this issue for later study.

The sixth question looks for specific bias in the study analysis. If such bias exists and if the bias cannot be compensated for, the results of the study are untrustworthy. Bias can be introduced by, for example, dropouts and noncompliance; thus, for clinical trials in medicine, it is common to demand an intention-to-treat analysis, in which each participant is analyzed in the group to which they were randomized regardless of what happened after randomization (see, e. g., Gupta 2011; Straus et al. 2011). Again, this question could (and should) be explicated further in later study.

It is quite possible that no study found in Step 2 will pass this quality appraisal step. At that point, there are basically three possibilities. First, the designer could review their quality appraisal to see if they were perhaps too harsh in what they counted as comparable and material. Second, the designer could go back to Step 2 and widen the search, if it was initially quite strict—for example, if the previous search was limited to randomized controlled trials, it may be a good idea to check out the nonrandomized controlled trials. Third, the designer may conclude (particularly if neither of the two strategies I just suggested is successful) that there simply isn't enough relevant quality research out there to carry out the evidence-based process.

10.2.4 Step 4—Applying the evidence

Once studies have been found and critically appraised, their message needs to be extracted and then used to make a decision. Practical instructions for doing this depend on the form of the designers' question and on the types of studies found. For comparative questions with a conventional controlled experiment, one might start by extracting the statistically significant relevant comparisons or relevant confidence intervals and then combine these pieces of information using a rough Bayesian inference from personal priors.[2] However, as discussed in Section 4.5, a formal Bayesian analysis requires quite a bit of effort and is probably not called for unless support software is available. The quality of statistical reporting in the surviving studies may also become a problem.

The simplest case is that there is only one study that survived the quality appraisal. In that case, the key questions are, what the reported result is and how well it answers the designers' question. If there is more than one study, the problem becomes more difficult: it is necessary first to determine the result that the studies point to together. The best technique would be to apply meta-analytic statistical techniques—whether mainstream or Bayesian—to pool the studies, but this is often impractical, particularly since designers may not have sufficient statistical expertise and experience; similarly impractical may often be applying the techniques discussed in Section 4.5 (unless support software is available). Sometimes it maybe possible to devise a theory that explains all the studies despite their divergent results—for example, a dissenting study may be methodologically weaker than other studies that agree on the outcome, or its result may be less relevant to the designer's problem.

However, it is not enough just to determine the outcome dictated by the studies. Straus et al. (2011, p. 1) describe EBM to

> "require[] the integration of the best research evidence with our clinical expertise and our patient's unique values and circumstances."

Similarly for EB-PLD: the final decision to choose a language design option depends not only on the research outcome but also on the designer's goals, their estimation of the preferences of the expected users of the language, among other things. For instance, the more different a language is compared to what the programmer is used to, the more costly adopting the language is likely to be for that programmer, and so novel ideas may require stronger evidence than common ones. The designer must weigh all relevant factors and make their decision accordingly.

10.2.5 Step 5—Evaluating one's own performance

The final step in the five-step EBx process is to evaluate one's performance after having gone through the first four steps. In medicine, Straus et al. (2011, Ch. 8)

[2] This preference for a Bayesian approach follows directly from adopting the inductive logic of Chapter 4 as the fundament of EB-PLD.

suggests that one reflect on each step: for example, they suggest reflecting on questions such as "[a]m I asking focused questions" (p. 206), "[a]m I becoming more efficient in my searching" (p. 206), "[a]m I critically appraising external evidence al all", and "[a]m I integrating my critical appraisals into my practice at all". Further, they suggest reflecting on whether using this process is improving one's practice.

While reflecting on one's skill in applying the five-step model may be useful even in EB-PLD, much more critical at this point is to ask the following questions after trying the process out:

1. Is EB-PLD practical for me?
2. Is EB-PLD providing any useful input to my decision-making?
3. Is EB-PLD worth the effort for me?

10.3 Thought experiments

I will now present two thought experiments involving fictional language designers applying my five-step process of EB-PLD. In essence, I will be role-playing the designers. My purpose here is to demonstrate how the process can be carried out in practice. They also argue for two propositions:

1. The EB-PLD process is feasible in principle.
2. The usefulness of EB-PLD depends heavily on the quantity and quality of existing studies.

The first thought experiment I wrote largely from top to bottom, first specifying the designer and their design problem and then following the process where it leads; I wrote it concurrently with writing the explication in the previous section, and writing it influenced the explication. While it does show the feasibility of the process, the result was disappointing: the studies do not actually help the designer much.

The second thought experiment was written after the explication was essentially complete. I also gave the process an advantage: I initially only chose the broad topic and completed Steps 2 and 3 to a large degree, and specified the designers and their question only after I had a fairly good idea of what the literature was capable of answering. Again, the process is shown feasible, and it also answers the designers' problem. It is not, however, a severe test of the process.

10.3.1 Syntax for conditionals

Suppose that a language designer were, for some good reasons that are irrelevant here, designing an improved C-like language for low-level system programming at a large, well estabished embedded systems company. Suppose further that the designer is not trying to preserve source code compatibility with C at all costs though breaking it without good reason is not acceptable. Now, suppose the

designer is unsure as to whether the existing syntax for conditionals is the best possible option available.

In Step 1, the designer formulates a question. The design decision is clear enough: the syntax of conditional statements is a good abstraction of the design problem which is found in many other language designs as well, and it is also already specified using common names. The evaluation criteria are, however, in this scenario completely unspecified and need to be identified. Their choice depends on the goals of the language designer; for the purposes of this hypothetical, I will assume that the criterion is the rate of logic errors made by professional programmers. My designer has no specific alternatives in consideration; thus, the form of the question will be what, not which. The resulting question is, therefore, the following:

> What syntax of the conditional statement is associated with the smallest rate of logic errors made by professional programmers?

In Step 2, the designer searchces for published evidence that might answer the question. They start by looking in the results of my Licentiate Thesis, reproduced in Chapter 8. Table 10 identifies three studies that investigate conditionals: S59 (Gilmore and Green 1984), S63 (Halverson 1993), and S79 (Iselin 1988). The next step is to cross-reference Table 11: since S59 and S79 use program comprehension instead of logic errors as their evaluation criterion, they are eliminated from further consideration. The remaining study, S63 (Halverson 1993), will be selected for critical appraisal.

The designer should also supplement this result with a new search looking for studies published after 2012, which was the cutoff point of the Licentiate Thesis. One possible supplemental search is to use Google Scholar with the search string

```
("conditional statement" OR "conditional statements" OR "if statement" OR
"if statements") ("logic error" OR "logic errors" OR "bugs") ("randomized
controlled experiment" OR "randomized controlled trial")
```

limiting the search to years starting from 2013; such a search conducted on July 28, 2015, yielded no results.

In Step 3, the designer applies my tentative quality appraisal checklist of Table 19 on p 168. The first question asks whether the manner in which the designs compared by the study differ is comparable to the design decision the designer is asking about. S63 (Halverson 1993) compares a single design of conditional statements to decision tables, but the designer is interested in differences in conditional statement syntax. S63 therefore fails the first item of the checklist and is excluded.

Since no other studies are left, the designer must return to Step 2 to perform another round of searching, this time for other experiments. Tables 12 and 14 identify eight additional studies that investigate conditionals—S27 (Chen and Vecchio 1992), S43 (Embley 1978), S60 (Green 1977), S136 (Shneiderman 1976; Shneiderman and Mayer 1979), S137 (Sime et al. 1973, 1999), S138 (Sime et al. 1977), S140 (Smith and Dunsmore 1982), and S151 (Vessey and Weber 1984a).

Cross-referencing Tables 13 and 15 shows that five of them—S27, S43, S60, S136, and S140—use program comprehension as their criterion of comparison, which leads to their exclusion. The remaining three—S137, S138, and S151—use error proneness as one of their criteria. These studies are therefore selected for critical appraisal.

The new search should also be supplemented. Google Scholar with the search string

```
("conditional statement" OR "conditional statements" OR "if statement" OR
"if statements") ("logic error" OR "logic errors" OR "bugs") ("controlled
experiment" OR "controlled trial")
```

limiting the search to years starting from 2013; such a search conducted on July 28, 2015, yielded the 46 results listed in Table 20. Of the results, #27 (Stefik and Siebert 2013) appears potentially relevant; a quick perusal shows that it reports, among other things, randomized controlled experiments to test e. g., conditionals for accuracy. It therefore is selected for appraisal. Dropping the word "controlled" from these searches yields 921 hits; since they are not conducting a systematic secondary study, there is no requirement on the designer to go through such an extensive search result set.

Now, the designer goes back to Step 3 and my tentative quality appraisal checklist of Table 19 on p 168. To pass the first question, studies should be comparing language designs whose only difference is with regard to their conditional statement syntax. The three mapping-study studies—S137 (Sime et al. 1973, 1999), S138 (Sime et al. 1977), and S151 (Vessey and Weber 1984a)—all compare combinations of the Sime languages JUMP, NEST, NEST-BE, and NEST-INE (cf. p. 33ff.). My hypothetical language designer believes that any solution reminiscent of JUMP—that is, forcing the programmer to write conditionals as jumps—is going to be heavily resisted by professional programmers, and thus they will not even consider JUMP as a viable language design. Similarly, NEST, which allows at most one substatement for each branch, is unworkable. The language designer therefore considers S137 (Sime et al. 1973, 1999) irrelevant and examines S138 (Sime et al. 1977) and S151 (Vessey and Weber 1984a) to the extent that they give evidence for preferring NEST-INE over NEST-BE, or vice versa. Therefore, S137 fails and the other two pass this item.

The study found by the additional search, #27 (Stefik and Siebert 2013), compares several existing languages and isolates features to study by using targeted tasks. So far as I can tell, they do not report results for isolated features, and thus this study fails this item and will not be considered further.

The second question of the quality checklist focuses on the criteria of comparison. For the hypothetical designer, the evaluation criterion specified in the question is the "rate of logic errors made by professional programmers". Both remaining studies—S138 (Sime et al. 1977), and S151 (Vessey and Weber 1984a)—compare error rates by people who have never (or almost never) programmed before. Thus, the criterion used by these studies is not identical to that used by my hypothetical designer, and the subjective element of "comparable" comes to fore. Does my hypothetical language designer believe that non-programmers are

TABLE 20 Titles of search results

1. An efficient regression testing approach for PHP web applications: a controlled experiment
2. An eye-tracking study assessing the comprehension of C++ and Python source code
3. Declarative visitors to ease fine-grained source code mining with full history on billions of AST nodes
4. Automatically generated patches as debugging aids: a human study
5. Automated Unit Test Generation during Software Development: A Controlled Experiment and Think-Aloud Observations
6. Toddler: Detecting performance problems via similar memory-access patterns
7. On the effect of code regularity on comprehension
8. Feature maintenance with emergent interfaces
9. A catalogue of refactorings to remove incomplete annotations
10. Mining interprocedural, data-oriented usage patterns in JavaScript web applications
11. Studying Fine-Grained Co-Evolution Patterns of Production and Test Code
12. Interactive code review for systematic changes
13. High-MCC functions in the Linux kernel
14. Stage-aware anomaly detection through tracking log points
15. Does automated white-box test generation really help software testers?
16. Bringing ultra-large-scale software repository mining to the masses with Boa
17. Comparing the maintainability of Selenium WebDriver test suites employing different locators: a case study
18. Bayesian methods: A social and behavioral sciences approach
19. Testing techniques selection based on ODC fault types and software metrics
20. Metallaxis-FL: mutation-based fault localization
21. Assessing the capability of code smells to explain maintenance problems: an empirical study combining quantitative and qualitative data
22. Understanding computer graphics student problem-solving through source-code analysis
23. Automatic Inference of Search Patterns for Taint-Style Vulnerabilitiesq
24. Determining the effectiveness of three software evaluation techniques through informal aggregation
25. Effectiveness of TDD on Unit Testing Practice
26. Assessing and improving quality in QVTo model transformations
27. An empirical investigation into programming language syntax (Stefik and Siebert 2013)
28. User-centered Program Analysis Tools
29. Mutation-oriented test data augmentation for GUI software fault localization
30. Protocol Programmability
31. MuRanker: a mutant ranking tool
32. Does automated unit test generation really help software testers? a controlled empirical study
33. Standard-compliant testing for safety-related automotive software
34. Supporting requirements update during software evolution
35. EMPIRICAL ASSESSMENT OF THE IMPACT OF USING AUTOMATIC STATIC ANALYSIS ON CODE QUALITY
36. Emergent Interfaces for Feature Modularization
37. SEMANTIC-AWARE ANOMALY DETECTION IN DISTRIBUTED STORAGE SYSTEMS.
38. How to measure anything: Finding the value of intangibles in business
39. Test-Driven Reuse: Improving the Selection of Semantically Relevant Code
40. Aspect-oriented programming evaluated: A study on the impact that aspect-oriented programming can have on software development productivity
41. Enhancing Learnability of Pair Programming Practice when Introducing Novices
42. Identification of Events in Use Cases
43. The Package Blueprint: Visually analyzing and quantifying packages dependencies
44. Impacts and Detection of Design Smells
45. Search-based Software Engineering
46. On Software Testing and Subsuming Mutants: An empirical study

comparable enough to professionals? One designer might exclude these studies on this basis, another might not be concerned at all. My hypothetical designer is aware of the problem and will take it into account when interpreting the results but they are not concerned enough to exclude these studies outright.

The third quality appraisal question involves a fair start. Consider now the two studies under consideration by my hypothetical designer.

- S138 (Sime et al. 1977) assigned one third of all participants to each of the three languages; no participant apparently tried more than one language. It is not reported how the assignment was accomplished.
- S151 (Vessey and Weber 1984a) had each participant learn and apply one of four languages (I will disregard here the other factor of indentation). Assignment was not random, nor was it completely under experimenters' control. Unlike the other studies, this study reports on the resulting balance of the groups and is frank about the notable imbalances that are present.

It remains to decide the fate of these studies. Both made some efforts to ensure and, disregarding the problems of participant assignment, do appear to give equal opportunity. However, both fail to either report or to handle participant assignment properly, which does cast significant doubt on equal opportunity. The decision whether to consider this a material problem or not is a judgment call; my hypothetical designer passes these studies with reservations, which they will take into account when interpreting the results.

As to the fourth question about fair race, none of the remaining studies being considered by my hypothetical designer show any worrisome signs to me, and thus all are given a tentative pass.

The fifth and sixth questions are about a fair finish. As to the fifth question, none of the remaining studies appear to contain serious errors in data analysis. As to the sixth question:

- S138 (Sime et al. 1977) compares the languages for differences in syntactic, semantic, and all errors. Participants are analyzed in the group of the language they actually used. No reason to exclude this study is apparent.
- S151 (Vessey and Weber 1984a) conducts a fairly complicated analysis from which it is difficult to determine whether and how the investigators analyzed the effect of language on error rates. It is my impression that this effect was not analyzed. Not analyzing at all is a subspecies of incorrect analysis, and therefore this study is excluded. This is not a statement on the overall quality of the study, just that it turns out to be irrelevant for my hypothetical designer.

Both the studies also measure and analyze other things, which are not relevant to my hypothetical designer's question.

The next step is Step 4, that of applying the evidence. Recall that my hypothetical designer is working on an improved C-like language for low-level system programming at a large, well estabished embedded systems company and is currently considering the syntax of conditional statements. The current design is

that inherited from C, but the designer is wondering if there are better designs available. Thus, they asked following question: "What syntax of the conditional statement is associated with the smallest rate of logic errors made by professional programmers?" The previous steps of the EB-PLD process yielded the study S138 (Sime et al. 1977), which has the noted quality issue that the researchers used non-programmers, not professional programmers, as their participants and neither report on how they assigned participants to languages.

The designer notes that the difference between NEST-BE and NEST-INE is readily translatable into his language, and they therefore consider it reasonably likely that any observed performance difference likewise transfer. Now, S138 (Sime et al. 1977) reports NEST-INE as having a median rate of 4 % semantic errors and 13 % syntactic errors, and NEST-BE as having a median rate of 7 % semantic and 85 % syntactic errors. It is not quite clear what the reference class here is, but I assume these to be the ratios of erroneous solutions to all solutions, normalized so that each person counts once. The difference between the three languages among each error category is reported to be statistically significant, but it is not reported whether the difference between NEST-INE and NEST-BE specifically is. However, the differences are not very small. It should be noted that NEST-BE underperforms JUMP for syntactic errors.

Since S138 (Sime et al. 1977) does not report pairwise significance tests nor does it specify confidence intervals for the medians, it is useless to try to infer any sort of probabilities using Bayesian inference or otherwise. However, the difference in syntactic errors is quite dramatic. My hypothetical designer concludes that this study shows that NEST-INE is superior to NEST-BE.

However, that conclusion may not mean that the designer should modify their language. As previously noted, that study did not report how participants were assigned to languages, and so there may not have been an equal opportunity to succeed; also, the participants were not professional programmers. It should be noted, as well, that the study is very old and that the programming interface used is very unusual (even by contemporary standards). Accordingly, the argument provided by this study is counted by my hypothetical designer as fairly weak; without doing detailed computations, they estimate the argument strength for NEST-INE being superior to NEST-BE to be somewhere close to .6, far from compelling.

Other arguments bear against changing the design. Programmers used to the C syntax (which is very common in other languages, as well) are likely to resist such a change. Modern programming environments may also provide much of the advantages of NEST-INE to a language applying NEST-BE. Lacking a compelling case, the designer decides against making any such change to their language.

In Step 5, the designer finds the method practical, but is disappointed that the existing evidence base did not present a compelling case. The input provided was somewhat useful even though it did not lead to a language change.

10.3.2 Software transactional memory

Consider now a pair of designers who have been collaborating on a design for a yet another Java-like language for writing server software. The language will be used by a company that sees a large turnaround in its programming staff, with new programmers being typically fresh graduates. They have been arguing for weeks about whether to base the language's concurrency facilities on Java-like monitors or on software transactional memory (STM); Barry claims that STM brings no advantages and only complicates implementation, while Sephie concedes the complications but posits savings in programming time. Finally, they have decided to try the EB-PLD process to see if the existing empirical literature can resolve their disagreement.

In Step 1, they formulate a question. It is straightforward:

> Does software transactional memory result in faster development time compared to monitors in a Java-like language used by recent graduates?

In Step 2, Barry and Sephie start from the mapping study, whose results are included in this dissertation. Table 10 (p. 134) shows three studies regarding software transactional memory—S22 (Castor, Oliveira, et al. 2011), S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011), and S127 (Rossbach et al. 2009, 2010). Cross-referencing Table 11 (p. 135) shows that all three use programming effort as one of their criteria of evaluation. Further, Table 12 (p. 136) shows an additional study—S94 (Luff 2009)—about software transactional memory; cross-referencing Table 13 (p. 137) shows that it uses proramming effort as one of its evaluation criteria.

Barry and Sephie supplement the search with a Google Scholar search

```
("software transactional memory") ("controlled experiment" OR "controlled
trial")
```

—there appearing to be no reasonable set of keywords to limit the search to just programming effort—performed on July 28, 2015; it yields the following four hits:

1. A preliminary assessment of Haskell's software transactional memory constructs (Castor, Soares-Neto, et al. 2013)
2. The programming language wars: Questions and responsibilities for the programming language community
3. Rewriting Concurrent Haskell programs to STM (Silva Neto 2014)
4. Communication for programmability and performance on multi-core processors (Luff 2013)

Based on a quick perusal of these publications, all except for one appear to report controlled experiments on software transactional memory, though #4 (Luff 2013) appears to only re-report S94 (Luff 2009), and #1 (Castor, Soares-Neto, et al. 2013), and #3 (Silva Neto 2014) appear to report the same study.

For Step 3, quality appraisal, Barry and Sephie therefore select the studies S22 (Castor, Oliveira, et al. 2011), S94/#4 (Luff 2009, 2013), S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011), S127 (Rossbach et

al. 2009, 2010), and #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014). They proceed according to the quality appraisal checklist in Table 19 (p. 168).

Checklist item 1 asks about whether the difference in the language designs studied is comparable to the design decision under question. S94/#4 (Luff 2009, 2013) and S127 (Rossbach et al. 2009, 2010) compare an implementation of STM in Java with Java monitors, and they therefore pass this item without issues. S22 (Castor, Oliveira, et al. 2011) and #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014) compare TVars to MVars in Haskell, by forbidding the group assigned to use one to use the other; TVars forcing the use of STM and MVars being distantly similar to Java monitors (although there are important differences), Barry and Sephie accept them as passing this item. S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) compares an implementation of STM with pthreads locking, which is about as similar to Java monitors as MVars are; it passess as well.

Checklist item 2 asks whether the evaluation criteria used in a study is comparable to that which is required in the question. All studies under consideration here do use programming time as one of their evaluation criteria. All of them use undergraduate (and in some cases graduate) students of computing as their participants. Barry and Sephie accept all of them as being comparable.

Checklist item 3 asks about a materially equal opportunity to succeed. S22 (Castor, Oliveira, et al. 2011), S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011), and #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014) all report assigning participants to groups randomly. In S22 (Castor, Oliveira, et al. 2011) and #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014), the participants were taught to use both TVars and MVars before the study, but it is not clear to if and to what extent potential differences in how they are taught jeopardize the required equal opportunity. S94/#4 (Luff 2009, 2013) does not report the method used to assign participants to groups, and S127 (Rossbach et al. 2009, 2010) reports using an assignment method which it claims to be random but is in fact systematic (assignment is determined by a function of a participant's name); in neither case is there any overt reason to suspect nonequal opportunity, but neither is there assurance of equal opportunity. Accordingly, only S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) passes this item without problems; Barry and Sephie will accept the others with reservations.

Checklist item 4 asks whether the studies treat the designs materially equally. Barry and Sephie detect no apparent problems in any of the studies under discussion and therefore they pass all of them without reservations.

Checklist item 5 asks whether there are serious problems in data analysis generally. S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) reports what is quite clearly an experiment (the investigators control the situation as instructors and, e. g., assign participants randomly to teams) and not a case study (in which the investigators would merely observe activity outside their control);[3] however, the study is explicitly guided by methodological

[3] Yin (2009, p. 11–12, emphasis added): "The case study is preferred [...] when the relevant

literature about case studies instead of experiments. Barry and Sephie (and I) are worried that this misidentification of the method may have caused unnoticed issues in the data collection and analysis, but they are not prepared to exclude this study on this basis alone.

Continuing with checklist item 5, Barry and Sephie accidentally notice that S22 (Castor, Oliveira, et al. 2011) reports incorrect data, in that the summary statistics in its Table 2 are inconsistent with the raw data in its Table 1. A systematic secondary study should probably contact the authors to resolve the problem, but Barry and Sephie are not conducting one; instead, they conclude that this study is fatally flawed in its reportage and fail it. S127 (Rossbach et al. 2009, 2010) in Table 2 of both publications reports 243 individual Wilcoxon signed-rank significance tests using $p < .05$ as the limit of statistical significance, of which 69 actually reach statistical significance. If no effects were to exist and all tests are independent, there is a probability of $1 - (1 - .05)^{243} \approx 1.00$ of at least one of them achieving $p < .05$ statistical significance and $243 \times .05 \approx 12$ would be expected to reach statistical significance (see, e. g., Bender and Lange 2001). Accordingly, while such multiple testing is suspicious in general, it is likely not a major problem here. However, it casts doubt into the methodological correctness of the rest of the analysis, which Barry and Sephie take into account. Similarly, #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014) conducts nine Student's t-tests on various variables collected from a single experiment; if no effects exist and the tests were independent (which they very likely are not), there is a probability of $1 - (1 - .05)^9 \approx .37$ of at least one of them achieving $p < .05$ statistical significance. Again, this casts doubt on methodological correctness. The studies thus far mentioned all pass with reservations; none of the other studies have apparent issues with respect to item 5, and thus pass without reservations.

Checklist item 6 asks about the material correctness of the specific analysis used to support the conclusion relevant to the designers' question.

- S94/#4 (Luff 2009, 2013) reports a moderate drop-out rate (6 out of 17) but does not specify how the drop-outs have been handled in the analysis. In reporting programmer time measurements, Luff uses the phrase "no significant difference", which suggests that he used some sort of significance test, but he gives no details; neither does he report descriptive statistics of the sample. However, the reported conclusion is plausible from the graphs he does include in his reports. Accordingly, this study passes with reservations.
- S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) appears to have had no drop-outs; all participants are included in the analysis. They do not report any statistical inference, though this is probably justified by the small number of participants. Accordingly, this study passes without reservations.

---

behaviors *cannot* be manipulated. [...E]xperiments are done when an investigator *can* manipulate behavior [..., e. g.,] in a field setting [...], where the investigators 'treat' whole groups of people in different ways, such as providing them with different kinds of voucers to purchase services".

– S127 (Rossbach et al. 2009, 2010) does not actually present any analysis of the overall question of time difference of STM and monitors and it thus must be excluded at this time.
– #1/#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014) reports using the two-tailed Student's t test, but there being multiple variants of that test, it is unclear which they mean. This study does not comment on dropouts. Accordingly, this study passes with reservations.

TABLE 21 Summary of the quality appraisal. An uppercase Y indicates passing, a lowercase y indicates passing with reservations, and an N indicates failing, which leads to the exclusion of the study from further consideration. The summary score counts the number of uppercase Ys.

Study	Checklist item						Summary
	1	2	3	4	5	6	score
S22	Y	Y	y	Y	N		
S94/#4	Y	Y	y	Y	Y	y	4
S114	Y	Y	Y	Y	y	Y	5
S127	Y	Y	y	Y	Y	N	
#1/#3	Y	Y	y	Y	y	y	3

This quality appraisal is summarized in Table 21. It also scores each study by counting the number of items it passes without reservations. This score must not be viewed as an adequate measure of quality (for one thing, it ignores the relative severity of the reservations) but it does indicate roughly where each study stands.

In Step 4, Barry and Sephie first extract the result of each study.

– S94/#4 (Luff 2009, 2013) does not report excact numbers but states that there is "no significant difference" (Luff 2009, p. 4). There were 17 participants initially, with 11 completing all tasks.
– S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011) reports a difference of 83 person-hours in total and a difference of 80 person-hours for implementation only between the time-sheet effort reported by three STM teams and three pthreads teams, in STM's favor. The pthread teams reported 693 person-hours in total and 348 person-hours for implementation only; thus, the difference is 12 % in total and 23 % for implementation only. No statistical inference is reported.
– #1#3 (Castor, Soares-Neto, et al. 2013; Silva Neto 2014) reports a difference of 14.92 minutes between time spent by MVar ($n = 32$) and TVar ($n = 24$) using groups, in TVar's favor. The MVar group spent an average of 143.68 minutes, and thus the difference was 10 %. They report a two-tailed Student's t-test giving $p = 0.0373$.

Barry and Sephie note that two of the three studies show a difference of about 10 %, which one of them concluded was statistically significant. That study was

also the largest of the three. The lone nonsignificant result is easily explained by the small number of participants. Accordingly, while they note the evidence is not particularly strong, Barry and Sephie unanimously conclude that there appears to be a real saving of programming effort in using software transactional memory. They agree to include it in their language. However, Barry insists that a lock-based alternative be also included in the language if it can be done without compromising on the STM's effectiveness, because the differences observed are not overwhelming enough to make him believe that STM is the best choice in all situations the language might be used.

In Step 5, Barry and Sephie note that the method was practical and gave useful input, and was worth the effort in this case.

Table 22 summarizes these two thought experiments.

10.4 Discussion

In this chapter, I have argued for a sketch of an evidence-based programming language design (EB-PLD) process and demonstrated it through two thought experiments. That EB-PLD ought to be explicated as the five-step process is, in my view, well supported. That there is a workable process to be discovered is strongly suggested by the two thought experiments. However, at its current stage of development, EB-PLD is not yet ready for its intended audience—practical language designers with no or little interest in research methodology.

The greatest problems are in Steps 3 and 4: performing a quality appraisal, and extracting data, requires at the present quite intricate understanding of research methods and particularly their pitfalls. More detailed treatment of these problems than that I have offered here requires, I believe, the development of a taxonomy of question forms and for each such, a separate checklist for quality appraisal and a procedure for data extraction. For the latter, off-the-shelf software, which may also be useful for other users of research, perhaps can be developed. These are, however, matters for future research.

Another problem comes from both the quantity and the quality of empirical research that could be used to fuel the process. For most real design problems, the research evidence simply does not exist. This cannot be solved by refining the EB-PLD method.

Looking back now to the Carnap (1962) criteria, I have argued, I think successfully, that this process is similar to what EB*x* should be when applied to programming language design. It can be fruitful in that it may provide reason to change or not change a language, but that depends mostly on the quality and quantity of the existing research. However, it is not yet exact or simple in the sense that it provides sufficient guidance to a language designer with no research

training. Accordingly, there is room for further research to improve on this explication.

TABLE 22 Summary of the two thought experiments

	Step	Section 10.3.1	Section 10.3.2
1	Formlating a question	What syntax of the conditional statement is associated with the smallest rate of logic errors made by professional programmers?	Does software transactional memory result in faster development time compared to monitors in a Java-like language used by recent graduates?
2	Locating evidence	The mapping study gives S60 (Green 1977), S137 (Sime et al. 1973, 1999), S138 (Sime et al. 1977), and S151 (Vessey and Weber 1984a). Google Scholar finds #27 (Stefik and Siebert 2013).	The mapping study gives S22 (Castor, Oliveira, et al. 2011), S94 (Luff 2009), S114 (Pankratius, Adl-Tabatabai, and Otto 2009; Pankratius and Adl-Tabatabai 2011), and S127 (Rossbach et al. 2009, 2010). Google Scholar finds #4 (Luff 2013), #1 (Castor, Soares-Neto, et al. 2013), and #3 (Silva Neto 2014). S94 and #4 report the same study; so do #1 and #3.
3	Appraising the evidence	S63 and S137 are excluded because the compared designs are not comparable to the design problem. #27 and S151 are excluded because the relevant comparisons are not analyzed in sufficient detail.	S22 is excluded because of doubts about the correctness of reported data. S127 is excluded because the relevant comparisons are not analyzed at all.
4	Applying the evidence	The remaining study S138 favors NESTINE to NEST-BE, which suggests a similar change to the language under development. Other considerations lead to not adopting the change.	S94/#4 reports a nonsignificant result, which can be explained by its small size. The two others, S114 and #1#3 report a similar clear effect, the latter concluding statistical significance. The studies therefore favor STM. The language under development will be equipped with STM.
5	Evaluating performance	Practical and somewhat useful but disappointing due to the paucity of evidence.	Practical and useful, and worth the effort in this case.

11 CONCLUSION

So, what does my iceberg (cf. p. 18) look like?

At the bottom, there is logic. There is the deductive logic, which I have assumed to be known. It is that which gives certain knowledge, which has zero information content about the real world. On top of that, there is the logic of induction, which allows concluding statements about the real world but with a cost: there is no longer any certainty. In fact, induction is fundamentally subjective, or so I have argued. Induction gives meaning to evidence, which consists basically of observations that drive induction. In particular, evidence includes a body of research reports.

Looking now at the top, there is a practical (though so far only sketched) process for resolving uncertainty in language design, Evidence-Based Programming Language Design. Below it, near the waterline, begins the body of research evidence. That body is supported by the concept of evidence, which brings us back to induction and logic. The theory of induction also provides a model for how study results are to be evaluated in preparation of a decision during the EB-PLD process, even though explicit computations might not be carried out.

Let me now review my research questions and give answers to them.

RQ1 *What should evidence-based programming language design (EB-PLD) consist of?*

I will start by restating Analysis 35:

> *Evidence-based programming language design* (*EB-PLD*) should be
>
> 1. a decision procedure to resolve uncertainty
> 2. regarding a particular practical language design problem
> 3. applied by an individual language designer
> 4. consisting of five steps:
>   (a) formulating a question
>   (b) locating evidence
>   (c) appraising the evidence
>   (d) applying the evidence
>   (e) evaluating one's own performance
>
> Here, *evidence* means relevant published studies.

Each of the items is important. If there is no uncertainty, there is nothing to do. If there is no actual design problem, there is nothing to do. If you are not the designer who is being uncertain about a design problem, there is nothing for you to do.

I have elaborated on each of the five steps at greater length in the subsections of Section 10.2, though that elaboration is left as a sketch. I will not repeat it here. I do, however, stress here that the process is subjective to the designer actually conducting it. I make no claim that this process will find an objectively correct answer that you could use to beat other language designers to death; in fact, I rather doubt any such process could be developed.

RQ2 *What does the word "evidence" mean?*

I uncovered two senses for this word that are to some extent in tension with each other. In EBM, evidence means published research reports; in my explication of EB-PLD, I have followed this approach. However, a *de novo* analysis reveals a much more expansive sense. To restate Analysis 8:

> *Evidence* is a report of observations offered as a premise in an argument whose conclusion is contingent, provided that the presence of this premise affects the strength of the argument.

This dichotomy between the *de novo* analysis and the one derived from EBM reinforces the conclusion that the phrase "evidence-based" is idiomatic and should not be analyzed from the meanings of its components.

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

The complete answer to this question encompasses the whole Chapter 8. Briefly, therefore: the body of evidence available for EB-PLD is rather small. In my mapping study, I found 141 studies, many of which were of marginal utility. The core comprises only 65 studies; of them, 22 were randomized controlled experiments. This data extends up to 2012.

RQ3 *Is Cohen's κ appropriate for assessing coder agreement in a systematic secondary study, and if not, what replacement should be used?*

Do *not* use Cohen's κ to assess coder agreement in a systematic secondary study. It deliberately discounts coder disagreement about the distribution of values. Acceptable replacements are, at least, Fleiss' κ, Scott's π, and Krippendorff's α. Do not confuse the last one with Cronbach's α, which is a completely different statistic.

I prefer Krippendorff's α. It is a family of statistics with a unified theory for all kinds of different data. In addition to nominal data, it can handle, among others, ordinal and interval data. In addition, it has no trouble dealing with missing values or more than two coders.

Now, this dissertation has clear limitations. I have already discussed the limitations of the mapping study in Section 8.3.2, and I will not say more about that. The rest of this dissertation is philosophical analysis, and that brings in some limitations.

First, as I already mentioned in the introduction, *dissent is to be expected*. By and large, my philosophical conclusions are supported by an inductive argument, which means that you commit no error if you agree with my premises but not with my conclusion. In addition, even where my argument is deductive, it is often possible for you to turn my *modus ponens* into your *modus tollens* by noting that since you disagree with my conclusion, there must be something wrong with my premises. My goal has been to give my arguments enough detail that if you are inclined to disagree, you can find purchase in my argument to construct a counterargument (and I hope you do so). If you agree, that's great; otherwise, let us advance the field by engaging in scholarly debate.

Second, my explication of EB-PLD has not been empirically evaluated. I do not dignify my hypothetical designer vingettes as empirical evaluation; their purpose is to give my discussion a concrete floor to walk on. Accordingly, I make no claim about its usefulness in practice. I do claim, however, that it warrants further study.

This brings me to open problems that I leave for further study. First, related to the further development of EB-PLD:

- a comprehensive taxonomy of question forms and evaluation criteria that can be used in EB-PLD
- an adequately justified hierarch of evidence
- empirical development and optimization of the EB-PLD search strategy
- a refined quality appraisal checklist
- quality appraisal checklists for non-comparative questions

Second, EB-PLD would need empirical validation at some point:

- Each step of EB-PLD needs to be evaluated separately for feasibility—non-comparative experiments would be one possible methodology.
- A partial or full EB-PLD could be simulated in classroom conditions in courses about the principles of programming languages.
- The full EB-PLD could be introduced in case studies or as action research in real-world language design projects.

I do not advocate running controlled experiments, because it is unclear what the control treatment could be. There are also the practical issue of how similar languages could be designed in parallel and independently of each other in such a way that experimental control is maintained.

These two categories of research problems highlight how this dissertation can viewed as a proposal for a Lakatosian research program. Lakatos (1970/1978) requires that each new theory in a research program come up with new predictions, although it is not required they survive severe testing immediately. The

chief prediction of EB-PLD at this point is that the process will allow a designer to resolve uncertainties, to the extent the literature contains relevant evidence; this prediction needs sharpening, of course, in later work. My first set of problems aim to develop the predictive power of EB-PLD, which would be the theory of this research program; the second set outline ways those predictions could be empirically tested.

There are also open research issues in other parts of this dissertation:

- Can a chance-corrected statistic of coder agreement be developed whose zero value indicates actual inability to distinguish observed from expected agreement, and only that (cf. Proposition 22 and the discussion following it)?
- The mapping study should be expanded to cover years after 2012 (preferably with improved methodology).
- A theory explaining how conclusions obtained from a comparative study of languages can be transferred to a new design would be very useful for both EB-PLD and for developing the methodology of comparative studies.

As always, more empirical studies that evaluate language design options are needed.

Returning to the standard of evaluation proposed by Dittrich (2015) (discussed on page 57), that the results should be relevant and the arguments rigorous, I have argued, I believe successfully, that my explication of evidence-based programming language design is relevant. Certainly, it has the potential to be useful once further refined. My arguments have been too extensive to review here in detail, but I believe I have given enough premises and small enough argumentative steps to be rigorous in that a dissenting reader can determine the central points of disagreement and generate counterarguments.

If you are going to call *something* evidence-based programming language design, it should be what I outlined in this dissertation. Perhaps such an approach is exactly what would allow an aspiring language designer to improve the state of the art. It is worth exploring.

YHTEENVETO (FINNISH SUMMARY)

Näyttöön perustuva ohjelmointikielten kehitys. Filosofinen ja menetelmäopillinen tutkimusmatka.

Väitöskirjassani pohdin, kuinka olemassa olevaa tutkimuskirjallisuutta voisi käyttää hyväksi ohjelmointikielten kehityksessä. Vastaavaan ongelmaan lääketieteessä kehitettiin 1970–1980-luvuilla lähestymistapa, jota 1990-luvun alusta on kutsuttu näyttöön perustuvaksi lääketieteeksi. Niinpä väitöskirjassani selvitän vastaavan *näyttöön perustuvan ohjelmointikielten kehityksen* mahdollista olemusta.

Väitöskirjan tutkimusmenetelmänä on toisaalta filosofinen käsiteanalyysi ja toisaalta systemaattinen kirjallisuuskartoitus. Käsiteanalyysillä selvensin näyttöön perustuvan lääketieteen olemusta ja sen siirtämistä soveltuvin osin ohjelmointikielten kehitykseen. Systemaattisella kirjallisuuskartoituksella, jonka aiemmin raportoin lisensiaattityönäni, selvitin, mitä empiiristä tutkimusnäyttöä on olemassa, jota voisi ohjelmointikielten näyttöön perustuvassa kehityksessä käyttää hyväksi. Lisäksi väitöskirja sisältää laajahkon tällaisten tutkimusten metodologiaa pohtivan osuuden.

Osana metodologian tarkastelua olen väitöskirjassani tarkentanut systemaattisten kirjallisuuskartoitusten ja -katsausten menetelmää kysyen, soveltuuko Cohenin kappa aineistokoodauksen luotettavuuden arviointiin. Matemaattisilla menetelmillä vastaukseksi selvisi ei. Sen asemesta suosittelen käyttämään Krippendorffin alfaa tai Fleissin kappaa.

Empiirisen tutkimusnäytön kirjallisuuskartoituksessa, jonka aineisto ulottuu vuoteen 2012 asti, havaitsin, että ohjelmointikielten suunnitteluratkaisujen hyödyllisyyttä on tutkittu jonkin verran: kaiken kaikkiaan tutkimuksia löytyi 141, ja näistä 65 tutkimusta, jotka ovat selkeimmin hyödyllisiä kielen kehittäjille, muodostavat tulosten ytimen. 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 kar-

toituksen 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 myös mahdollista, että julkaisufoorumien toimittajat ja vertaisarvioijat pitävät empiiristä tutkimusta niin hyödyttömänä, ettei sellaisia tutkimuksia juuri niiden tekemisestä huolimatta julkaista. On toki 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.

Väitöskirjan tuloksena syntynyt näyttöön perustuvan ohjelmointikielten kehityksen hahmotelma on seuraava. Jos kielen kehittäjälle on tosiasiassa epäselvää, miten jokin kehitysongelma tulisi ratkaista, hän voi alkaa soveltaa menetelmän viisiaskelista toimintaohjetta. Ensin hän täsmentää kysymyksensä, sitten hän tekee kysymyksen perusteella kirjallisuushaut, kolmanneksi hän arvioi löytyneiden lähteiden luotettavuuden, neljänneksi soveltaa kirjallisuuden antamaa ratkaisua ongelmaansa ja viidenneksi pohtii, kuinka menetelmän soveltaminen häneltä onnistui. Tämä toimintaohje on tällä yleistasolla hyvin samanlainen kuin näyttöön perustuvassa lääketieteessä; eroja tulee, kun askelille annetaan tarkemmat toteutusohjeet.

Kuten kaikilla tutkimuksilla, tällä väitöskirjalla on rajoitteita, jotka tulee tuloksia tulkittaessa ottaa huomioon. Käsiteanalyysin tärkein rajoite on, etteivät sillä tavalla syntyvät tulokset ole todistettavissa oikeiksi; paraskaan käsiteanalyysi ei välttämättä vakuuta kaikkia lukijoita. Väitöstyöni käsiteanalyysin tavoitteena on siten ollut, että eri mieltä oleva lukija kykenee vastaamaan argumentteihini järkevillä vasta-argumenteilla ja siten edistämään aihetta koskevaa tieteellistä keskustelua. Kirjallisuuskartoituksen keskeisin rajoite puolestaan 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.

Väitöskirjan olennaisin tulos on näyttöön perustuvan ohjelmointikielten kehityksen hahmotelma, joka soveltuu jatkotutkimuksen pohjaksi. Sitä ei ole empiirisesti arvioitu, joten lopullisena vastauksena sitä ei voi pitää. Sen käytettävissä nykyisin oleva empiirinen tutkimusnäyttö on ohutta, mutta sitä on.

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. 23).

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 p. 33).

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. 116).

ACM s.d.(b). Zone and Field Searches. ⟨URL: http://dl.acm.org/documentation/Zone.htm⟩ [visited on 2011-10-19] (cit. on p. 116).

Aczél, J. 2004. The Associativity Equation Re-Revisited. AIP Conference Proceedings 707 (3), 195–203. DOI: 10.1063/1.1751367 (cit. on p. 64).

Adams, M. D. 2015. Towards the Essence of Hygiene. In POPL'15. Proceedings of the 42nd Annual ACM SIGPLAN–SIGACT Symposium on Principles of Programming Languages, 457–469. DOI: 10.1145/2676726.2677013 (cit. on p. 55).

Agerbo, E. and Cornils, A. 1998. How to Preserve the Benefits of Design Patterns. In Proceedings of the 13th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 134–143. DOI: 10.1145/286936.286952 (cit. on p. 56).

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 p. 131).

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. 33).

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 p. 131).

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 p. 149).

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 p. 131).

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. 30).

Allende, E., Callaú, O., Fabry, J., Tanger, É., and Denker, M. 2013. Gradual typing for Smalltalk. Science of Computer Programming 96, 52–69. DOI: 10.1016/j.scico.2013.06.006 (cit. on p. 36).

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. 27).

American Standard Code for Information Interchange 1963. Communications of the ACM 6 (8), 422–426. DOI: 10.1145/366707.367524 (cit. on p. 29).

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 p. 131).

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. 41, 131).

Aristotle 2014. Categories. In The Complete Works of Aristotle. The Revised Oxford Translation. Ed. by Barnes, J. Trans. by Ackrill, J. L. Digital edition. Princeton / Bollingen Series LXXI:2. Princeton University Press. ⟨URL: http://amzn.com/B00JW04P64⟩ (cit. on p. 48).

Arnauld, A. and Nicole, P. 1996. Logic or the Art of Thinking. Containing, besides common rules, several new observations appropriate for forming judgment. Ed. and trans. by Buroker, J. V. Fifth edition, revised and newly augmented. Cambridge texts in the history of philosophy. Cambridge University Press (cit. on p. 48).

Ash, R. B. 2008. Basic Probability Theory. Unabridged republication of a Wiley edition published in 1970. Mineola, NY: Dover. ⟨URL: http://www.math.uiuc.edu/~r-ash/BPT/BPT.pdf⟩ [visited on 2015-03-22] (cit. on p. 80).

Ayer, A. J. 2001. Language, Truth and Logic. London: Penguin. ⟨URL: https://play.google.com/store/books/details?id=3M3ycPeuRNoC⟩ (cit. on p. 16).

Babar, M. I., Ghazali, M., and Jawawi, D. N. A. 2014. Systematic Reviews in Requirements Engineering. A Systematic Review. In 8th Malaysian Software Engineering Conference (MySEC), 43–48. DOI: 10.1109/MySec.2014.6985987 (cit. on p. 157).

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. 32, 34).

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. 40).

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 p. 131).

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 p. 131).

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 p. 131).

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. 86).

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. 19).

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. 38).

Barends, E. G. R. and Briner, R. B. 2014. Teaching Evidence-Based Practice: Lessons from the Pioneers. An Interview With Amanda Burls and Gordon Guyatt. Academy of Management Learning & Education 13 (3), 476–483. DOI: 10.5465/amle.2014.0136 (cit. on p. 152).

Bar-Hillel, M. 1980. The Base-Rate Fallacy in Probability Judgments. Acta Psychologica 44 (3), 211–233. DOI: 10.1016/0001-6918(80)90046-3 (cit. on p. 74).

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 p. 131).

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 p. 131).

Beaney, M. 1997a. Introduction. In The Frege Reader. Ed. by Beaney, M. Oxford: Blackwell, 1–46 (cit. on p. 48).

Beaney, M., ed. The Frege Reader 1997b. Oxford: Blackwell.

Bell, K. 2012. Cochrane Reviews and the Behavioural Turn in Evidence-Based Medicine. Health Sociology Review 21 (3), 313–321. DOI: 10.5172/hesr.2012.21.3.313 (cit. on p. 154).

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. 161).

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 p. 131).

Ben-Ari, M. 2012. Mathematical Logic for Computer Science. 3rd ed. London: Springer. DOI: 10.1007/978-1-4471-4129-7 (cit. on p. 59).

Bender, R. and Lange, S. 2001. Adjusting for multiple testing. When and how? Journal of Clinical Epidemiology 54 (4), 343–349. DOI: 10.1016/S0895-4356(00)00314-0 (cit. on p. 179).

Benétreau-Dupin, Y. 2015. The Bayesian who knew too much. Synthese 192 (5), 1527–1542. DOI: 10.1007/s11229-014-0647-3 (cit. on p. 76).

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 p. 131).

Berger, J. 2006. Rejoinder. Bayesian Analysis 1 (3), 457–464. DOI: 10.1214/06-BA116REJA (cit. on p. 79).

Bergin Jr., T. J. and Gibson Jr., R. G., eds. History of Programming Languages—II 1996. New York: ACM Press (cit. on p. 40).

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. 84).

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 p. 131).

Billingsley, P. 1995. Probability and Measure. 3rd ed. New York: Wiley (cit. on p. 80).

Blackmore, J. 1979. On the Inverted Use of the Terms 'Realism' and 'Idealism' Among Scientists and Historians of Science. British Journal for the Philosopht of Science 30 (2), 125–134. ⟨URL: http://www.jstor.org/stable/686944⟩ (cit. on pp. 46, 60).

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. 23, 24).

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. 42).

Blunt, C. J. 2014. Hierarchies of Evidence. ⟨URL: http://cjblunt.com/hierarchies-evidence/⟩ [visited on 2015-04-17] (cit. on p. 160).

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 p. 131).

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 p. 131).

Borgerson, K. 2008. Valuing and Evaluating Evidence in Medicine. PhD thesis. University of Toronto. ⟨URL: http://hdl.handle.net/1807/11182⟩ [visited on 2015-04-28] (cit. on p. 160).

Borgerson, K. 2009. Valuing Evidence. Bias and Evidence Hierarchy of Evidence-Based Medicine. Perspectives in Biology and Medicine 52 (2), 218–233. DOI: 10.1353/pbm.0.0086 (cit. on p. 160).

Bos, J. van den and Storm, T. van der 2013. A Case Study in Evidence-Based DSL Evolution. In Modelling Foundations and Applications. 9th European Conference, ECMFA 2013, Montpellier, France, July 1-5, 2013, Proceedings. Ed. by Van Gorp, P., Ritter, T., and Rose, L. M. Lecture Notes in Computer Science 7949. Berlin: Springer, 207–219. DOI: 10.1007/978-3-642-39013-5_15 (cit. on p. 163).

Bourbaki, N. 2004. Elements of Mathematics. Theory of Sets. Berlin: Springer. DOI: 10.1007/978-3-642-59309-3 (cit. on p. 51).

Brendel, E. 2004. Intuition Pumps and the Proper Use of Thought Experiments. Dialectica 58 (1), 89–108. DOI: 10.1111/j.1746-8361.2004.tb00293.x (cit. on p. 52).

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 p. 131).

Briner, R. B., Denyer, D., and Rousseau, D. M. 2009. Evidence-Based Management. Concept Cleanup Time? Academy of Management Perspectives 23 (4), 19–32. ⟨URL: http://amp.aom.org/content/23/4/19.short⟩ (cit. on pp. 16, 155, 159).

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. 40).

Brown, N. C., Andreazza, A. C., and Young, L. T. 2014. An updated meta-analysis of oxidative stress markers in bipolar disorder. Psychiatry Research 218 (1–2), 61–68. DOI: 10.1016/j.psychres.2014.04.005 (cit. on p. 83).

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. 88).

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. 84).

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. 148).

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. 85, 88).

Bundy, A., Jamnik, M., and Fugard, A. 2005. What is a proof? Philosophical Transactions of the Royal Society. A: Mathematical, Physical & Engineering Sciences 363 (1835), 2377–2391. DOI: 10.1098/rsta.2005.1651 (cit. on p. 51).

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 p. 131).

Burgess, J. P. 2009. Philosophical Logic. Princeton University Press (cit. on p. 51).

Butchvarov, P. 2003. Ethics Dehumanized. Southern Journal of Philosophy 41 (S1), 165–183. DOI: 10.1111/j.2041-6962.2003.tb00985.x (cit. on p. 47).

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 p. 131).

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. 25).

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. 25).

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 p. 38).

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. 36, 38).

Carnap, R. 1956. Meaning and Necessity. 2nd ed. Chicago: University of Chicago Press (cit. on pp. 45, 49).

Carnap, R. 1962. Logical Foundations of Probability. 2nd ed. Chicago: University of Chicago Press (cit. on pp. 45, 47, 48, 54, 58, 164, 181).

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. 125, 131, 134, 135).

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 p. 131).

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. 131, 134, 135, 177–179, 183).

Castor, F., Soares-Neto, F., and Santos, A. L. M. 2013. A Preliminary Assessment of Haskell's Software Transactional Memory Constructs. In The 28th Annual ACM Symposium on Applied Computing, 1696–1697. DOI: 10.1145/2480362.2480681 (cit. on pp. 177–180, 183).

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. 87).

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 p. 131).

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. 131, 140).

Chalmers, I. 1999. Why transition from alternation to randomisation in clinical trials was made. BMJ 319 (7221), 1372. ⟨URL: http://europepmc.org/articles/PMC1117101⟩ (cit. on p. 160).

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. 84).

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 p. 131).

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 p. 131).

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. 131, 138, 172).

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. 86).

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. 131, 134, 135).

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. 88).

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. 38).

Church, A. 1936. An Unsolvable Problem of Elementary Number Theory. American Journal of Mathematics 58 (2), 345–363. ⟨URL: http://www.jstor.org/stable/2371045⟩ (cit. on p. 55).

Church, A. 1937. Review of A. M. Turing, On computable numbers, with an application to the Entscheidungsproblem. Journal of Symbolic Logic 2 (1), 42–43. ⟨URL: http://www.jstor.org/stable/2268810⟩ (cit. on p. 55).

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. 38).

Church, A. 1941. The Calculi of Lambda-Conversion. Annals of Mathematics Studies 6. Princeton University Press (cit. on p. 38).

Church, A. 1946. Review of Morton G. White, A Note on the "paradox of analysis"; Max Black, The "paradox of analysis" again: a reply; Morton G. White, Analysis and identity: a rejoinder; and Max Black, How can analysis be informative? Journal of Symbolic Logic 11 (4), 132–133. ⟨URL: http://www.jstor.org/stable/2268327⟩ (cit. on p. 48).

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. 37).

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. 37).

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. 35).

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 p. 131).

Cohen, A. M., Stavri, P. Z., and Hersh, W. R. 2004. A categorization and analysis of the criticisms of evidence-based medicine. International Journal of Medical Informatics 73 (1), 35–43. DOI: 10.1016/j.ijmedinf.2003.11.002 (cit. on pp. 154, 160).

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. 89, 95, 122).

Cohen, J. 1968. Weighted kappa. Nominal scale agreement with provision for scaled disagreement or partial credit. Psychological Bulletin 70 (4), 213–220 (cit. on p. 102).

Cohen, J. 1988. Statistical Power Analysis for the Behavioral Sciences. 2nd ed. Lawrence Erlbaum. ⟨URL: https://play.google.com/store/books/details?id=cIJH0lR33bgC⟩ (cit. on p. 73).

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. 131, 140).

Colburn, T. R. 2000. Philosophy and Computer Science. Armonk: Sharpe (cit. on p. 22).

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. 161).

Computer Science Curricula 2013. Curriculum Guidelines for Undergraduate Degree Programs in Computer Science 2013. Dec. 2013. DOI: 10.1145/2534860 (cit. on p. 28).

Computing Curricula 2001. Computer Science Final Report 2001. Dec. 2001. DOI: 10.1145/384274.384275 (cit. on p. 28).

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 p. 131).

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. 41, 42).

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. 41).

Cox, R. T. 1946. Probability, Frequency and Reasonable Expectation. American Journal of Physics 14 (1), 1–13. DOI: 10.1119/1.1990764 (cit. on pp. 59, 64, 68).

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. 26).

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. 90, 123–125).

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. 83, 90).

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. 91).

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 p. 131).

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. 35).

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. 131, 134, 135).

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. 131, 134, 135).

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. 131, 134, 135, 143).

Davidoff, F., Haynes, B., Sackett, D., and Smith, R. 1995. Evidence Based Medicine. A new journal to help doctors identify the information they need. BMJ 310, 1085–1086. DOI: 10.1136/bmj.310.6987.1085 (cit. on p. 154).

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. 27).

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, a1. DOI: 10.1186/1472-6920-5-1 (cit. on pp. 153, 159).

Dawid, A. P. 1987. The Difficulty About Conjunction. The Statistician 36 (2–3), 91–97. ⟨URL: http://www.jstor.org/stable/2348501⟩ (cit. on p. 76).

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 p. 131).

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 p. 131).

Dennett, D. 1980. The milk of human intentionality. Behavioral and Brain Sciences 3 (3), 428–430. DOI: 10.1017/S0140525X0000580X. ⟨URL: http://hdl.handle.net/10427/57583⟩ (cit. on p. 52).

Dennett, D. C. 2013. Intuition Pumps and Other Tools for Thinking. Penguin. ⟨URL: http://amzn.com/B00BQ4NIFI⟩ (cit. on p. 52).

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. 113).

Dershem, H. L. and Jipping, M. J. 1995. Programming Languages. Structures and Models. Boston: PWS (cit. on pp. 21, 22).

Détienne, F. 2002. Software Design. Cognitive Aspects. Trans. by Bott, F. Practitioner Series. London: Springer (cit. on pp. 23, 41, 42, 147).

DiCenso, A., Bayley, L., and Haynes, R. B. 2009. Accessing Pre-Appraised Evidence. Fine-tuning the 5S model into a 6S model. Evidence-Based Nursing 12 (4), 99–101. DOI: 10.1136/ebn.12.4.99-b (cit. on pp. 161, 166).

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. 86, 87).

Dietrich, E. 2011. There Is No Progress in Philosophy. Essays in Philosophy 12 (2), 9. ⟨URL: http://commons.pacificu.edu/eip/vol12/iss2/9/⟩ (cit. on p. 50).

Dittrich, Y. 2015. What does it mean to use a method? Towards a practice theory for software engineering. Information and Software Technology. DOI: 10.1016/j.infsof.2015.07.001. Pre-published (cit. on pp. 57, 187).

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. 131, 136, 137).

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. 131, 140).

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 p. 131).

Durrett, R. 2013. Probability. Theory and Examples. 4.1. Fourth edition was published by Cambridge University Press on 2010. Durrett. ⟨URL: http://www.math.duke.edu/~rtd/PTE/PTE4_1.pdf⟩ [visited on 2015-03-22] (cit. on p. 80).

Dybå, T. and Dingsøyr, T. 2008. Strength of Evidence in Systematic Reviews in Software Engineering. In ESEM'08. Proceedings of the 2008 ACM–IEEE International Symposium on Empirical Software Engineering and Measurement. DOI: 10.1145/1414004.1414034 (cit. on p. 167).

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 p. 147).

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. 131, 138).

Earman, J. 1992. Bayes or Bust? A Critical Examination of Bayesian Confirmation Theory. Cambridge, MA: MIT Press. ⟨URL: http://pitt.edu/~jearman/Earman_1992BayesOrBust.pdf⟩ [visited on 2015-03-19] (cit. on p. 71).

Easwaran, K. 2011a. Bayesianism I. Introduction and Arguments in Favor. Philosophy Compass 6 (5), 312–320. DOI: 10.1111/j.1747-9991.2011.00399.x (cit. on p. 68).

Easwaran, K. 2011b. Bayesianism II. Applications and Criticisms. Philosophy Compass 6 (5), 321–332. DOI: 10.1111/j.1747-9991.2011.00398.x (cit. on p. 68).

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 p. 131).

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. 33, 34, 131, 136, 137, 144, 172).

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. 33).

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 p. 131).

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. 131, 134, 135).

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 p. 131).

Evidence-Based Medicine Working Group 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. 16, 152, 153, 158, 159).

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. 22).

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 p. 131).

Feinstein, A. R. and Cicchetti, D. V. 1990. High Agreement but Low Kappa. I. The problems of two paradoxes. Journal of Clinical Epidemiology 43 (6), 543–549 (cit. on pp. 96, 97, 100, 105).

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. 89).

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. 89).

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. 91).

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. 91).

Felleisen, M. 1991. On the expressive power of programming languages. Science of Computer Programming 17 (1–3), 35–75. DOI: 10.1016/0167-6423(91)90036-W (cit. on p. 15).

Feng, G. C. 2013. Factors Affecting Intercoder Reliability. A Monte Carlo experiment. Quality & Quantity 47 (5), 2959–2982. DOI: 10.1007/s11135-012-9745-9 (cit. on p. 100).

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. 131, 139, 140).

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 p. 131).

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. 131, 138).

Fine, T. L. 1973. Theories of Probability. An Examination of Foundations. New York: Academic. ⟨URL: https://play.google.com/store/books/details?id=brDiBQAAQBAJ⟩ (cit. on pp. 59, 79).

Finetti, B. de 1992. Foresight. It's Logical Laws, Its Subjective Sources. In Breakthroughs in Statistics. Volume I: Foundations and Basic Theory. Ed. by Kotz, S. and Johnson, N. L. Trans. by Kyberg Jr., H. E. Springer. DOI: 10.1007/978-1-4612-0919-5_10 (cit. on p. 68).

Fisher, R. A. 1937. The Design of Experiments. 2nd ed. Edinburg: Oliver and Boyd (cit. on p. 161).

Fisher, R. A. 2005. Statistical Methods for Research Workers. Classics in the History of Psychology. Christopher D. Green. ⟨URL: http://psychclassics.yorku.ca/Fisher/Methods/⟩ (cit. on p. 98).

Fitting, M. 1996. First-Order Logic and Automated Theorem-Proving. New York: Springer (cit. on p. 59).

Fitzgerald, B., Musiał, M., and Stol, K.-J. 2014. Evidence-Based Decision Making in Lean Software Project Management. In 36th International Conference on Software Engineering (ICSE Companion 2014). Proceedings, 93–102. DOI: 10.1145/2591062.2591190 (cit. on p. 157).

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 p. 131).

Fleiss, J. L. 1971. Measuring Nominal Scale Agreement Among Many Raters. Psychological Bulletin 76 (5), 378–382 (cit. on pp. 104, 122).

Fleiss, J. L. and Cohen, J. 1973. The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability. Educational and Psychological Measurement 33 (3), 613–619 (cit. on p. 102).

Fleiss, J. L., Levin, B., and Paik, M. C. 2003. Statistical Methods for Rates and Proportions. 3rd ed. Hoboken, NJ: Wiley (cit. on p. 102).

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. 27, 29).

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. 32).

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. 131, 140).

Frege, G. 1948. Sense and Reference. Trans. by Black, M. Philosophical Review 57 (3), 209–230. DOI: 10.2307/2181485 (cit. on p. 48).

Frege, G. 1997. Function and Concept. In The Frege Reader. Ed. by Beaney, M. Trans. by Geach, P. Oxford: Blackwell, 130–148 (cit. on p. 49).

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. 40).

Frogel, S. 2005. The Rhetoric of Philosophy. Amsterdam: Benjamins (cit. on p. 53).

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 p. 131).

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 p. 131).

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 p. 131).

Gabbrielli, M. and Martini, S. 2010. Programming Languages. Principles and Paradigms. London: Springer. DOI: 10.1007/978-1-84882-914-5 (cit. on pp. 22, 31).

Gaifman, H. 1964. Concerning Measures in First Order Calculi. Israel Journal of Mathematics 2 (1), 1–18. DOI: 10.1007/BF02759729 (cit. on pp. 65, 71).

Gaifman, H. and Snir, M. 1982. Probabilities over Rich Languages, Testing and Randomness. Journal of Symbolic Logic 47 (3), 495–548. ⟨URL: http://www.jstor.org/stable/2273587⟩ (cit. on pp. 65, 71, 75).

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. 131, 136, 137, 144).

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 p. 131).

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 p. 131).

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 p. 131).

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. 83).

Gelman, A., Carlin, J. B., Stern, H. S., Dunson, D. B., Vehtari, A., and Rubin, D. B. 2013. Bayesian Data Analysis. 3rd ed. CRC. ⟨URL: https://play.google.com/store/books/details?id=eSHSBQAAQBAJ⟩ (cit. on pp. 72, 79).

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. 31).

Gettier, E. L. 1963. Is justified true belief knowledge? Analysis 23 (6), 121–123. DOI: 10.1093/analys/23.6.121. ⟨URL: http://www.jstor.org/stable/3326922⟩ (cit. on pp. 46, 52).

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. 23).

Ghahramani, Z. 2015. Probabilistic machine learning and artificial intelligence. Nature 521 (7553), 452–459. DOI: 10.1038/nature14541 (cit. on p. 79).

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. 131, 140).

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. 131, 140).

Gill, J. 2015. Bayesian Methods. A Social and Behavioral Sciences Approach. 3rd ed. CRC. ⟨URL: https://play.google.com/store/books/details?id=6fDRBQAAQBAJ⟩ (cit. on pp. 72, 79).

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. 131, 134, 135, 172).

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. 40).

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. 38).

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. 83).

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. 114).

Gödel, K. 1967. The completeness of the axioms of the functional calculus of logic. In From Frege to Gödel. A Source Book in Mathematical Logic, 1879–1931. Ed. by Heijenoort, J. van. Trans. by Bauer-Mengelberg, S. Cambridge, MA: Harvard University Press, 582–591 (cit. on p. 60).

Godlee, F. 2014. Evidence-Based Medicine. Flawed system but still the best we've got. BMJ 348, g440. DOI: 10.1136/bmj.g440 (cit. on p. 16).

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. 36).

Goldenberg, M. J. 2009. Iconoclast or Creed? Objectivism, Pragmatism, and the Hierarchy of Evidence. Perspectives in Biology and Medicine 52 (2), 168–187. DOI: 10.1353/pbm.0.0080 (cit. on p. 160).

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. 27).

Gomolińska, A. 1997. A Nonmonotonic Modal Formalization of the Logic of Acceptance and Rejection. Studia Logica 58 (1), 113–127. DOI: 10.1023/A:1004996032649 (cit. on p. 73).

Gomolińska, A. 1998. On the Logic of Acceptance and Rejection. Studia Logica 60 (2), 233–251. DOI: 10.1023/A:1005009115432 (cit. on p. 73).

Google 2011. Google Scholar Help. ⟨URL: http://scholar.google.com/intl/en/scholar/help.html⟩ [visited on 2011-09-09] (cit. on p. 149).

Gordon, M. J., Milner, A. J., and Wadsworth, C. P. 1979. Edinburgh LCF. A Mechanised Logic of Computation. Lecture Notes in Computer Science 78. Berlin: Springer. DOI: 10.1007/3-540-09724-4 (cit. on p. 38).

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. 22, 34).

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. 30).

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. 26).

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. 40).

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. 32).

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. 131, 138, 172, 183).

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 p. 131).

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. 42).

Greenhalgh, T., Howick, J., Maskrey, N., et al. 2014. Evidence based medicine. A movement in crisis? BMJ 348, g3725. DOI: 10.1136/bmj.g3725 (cit. on pp. 16, 154, 155).

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 p. 131).

Grier, D. A. 2007. When Computers Were Human. Princeton University Press (cit. on p. 55).

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. 31).

Grove, W. M., Andreasen, N. C., McDonald-Scott, P., Keller, M. B., and Shapiro, R. W. 1981. Reliability Studies of Psychiatric Diagnosis. Archives of General Psychiatry 38 (4), 408–413 (cit. on pp. 96, 97, 105).

Gupta, S. K. 2011. Intention-to-treat concept. A review. Perspectives in Clinical Research 2 (3), 109–112. DOI: 10.4103/2229-3485.83221 (cit. on p. 169).

Gwet, K. L. 2008. Computing Inter-Rater Reliability and its Variance in the Presence of High Agreement. British Journal of Mathematical and Statistical Psychology 61 (1), 29–48. DOI: 10.1348/000711006X126600 (cit. on pp. 97, 98, 104).

Gwet, K. L. 2012. Handbook of Inter-Rater Reliability. The Definitive Guide to Measuring the Extent of Agreement Among Multiple Raters. 3rd ed. Gaithersburg, MD: Advanced Analytics (cit. on pp. 17, 97, 98, 102).

Hacker, P. M. S. 2009. Philosophy: A Contribution, not to Human Knowledge, but to Human Understanding. Royal Institute of Philosophy Supplement 65, 129–153. DOI: 10.1017/S1358246109990087 (cit. on p. 54).

Hacker, P. M. S. 2013. The Linguistic Turn in Analytic Philosophy. In Oxford Handbook of the History of Analytic Philosophy. Oxford University Press. ⟨URL: http://info.sjc.ox.ac.uk/scr/hacker/docs/TheLinguisticTurn.pdf⟩ (cit. on pp. 46, 49).

Hacking, I. 1988. Telepathy. Origins of Randomization in Experimental Design. ISIS 79 (3), 427–451. ⟨URL: http://www.jstor.org/stable/234674⟩ (cit. on p. 160).

Hahn, U. and Oaksford, M. 2006. A Normative Theory of Argument Strength. Informal Logic 26 (1), 1–24. ⟨URL: http://ojs.uwindsor.ca/ojs/leddy/index.php/informal_logic/article/viewArticle/428⟩ [visited on 2015-06-02] (cit. on p. 59).

Halpern, J. Y. 1999. Cox's Theorem Revisited. Journal of Artificial Intelligence Research 11, 429–435. DOI: 10.1613/jair.644 (cit. on p. 80).

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. 22).

Halverson Jr., R. 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. 131, 134, 135, 143, 172).

Hamilton, W. 1861. Lectures on Metaphysics and Logic. 2nd ed. Vol. II. 4 vols. Edinburgh: William Blackwood. ⟨URL: https://play.google.com/store/books/details?id=OltEAAAAcAAJ⟩ (cit. on p. 48).

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 p. 131).

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. 131, 136, 137).

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. 117, 131, 136, 137).

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. 117, 131, 136, 137).

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. 42, 56).

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. 36).

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. 31).

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 p. 131).

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. 31, 36).

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. 131, 134, 135).

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 p. 131).

Harrison, W. and Ossher, H. 1993. Subject-Oriented Programming. A Critique of Pure Objects. In OOPSLA 93 Conference on Object Oriented Programming Systems Languages and Applications Washington, DC, USA — September 26 - October 01, 1993, 411–428. DOI: 10.1145/165854.165932 (cit. on p. 56).

Hart, P. D. 1999. A Change in Scientific Approach. From alternation to randomized allocation in clinical trials in the 1940s. BMJ 319 (7209), 572–573. ⟨URL: http://europepmc.org/articles/PMC1116443/⟩ (cit. on p. 160).

Hawking, S. and Mlodinow, L. 2010. The Grand Design. London: Transworld. ⟨URL: https://play.google.com/store/books/details?id=xKDnWSd5SC4C⟩ (cit. on p. 45).

Hawthorne, J. 2014. Bayesian Confirmation Theory. In The Bloomsbury Companion to the Philosophy of Science. Ed. by French, S. and Saatsi, J. New York: Bloomsbury. Chap. 11. ⟨URL: https://play.google.com/store/books/details?id=5HHjAwAAQBAJ⟩ (cit. on p. 79).

Haynes, R. B. 2001. Of studies, summaries, synopses, and systems. The "4S" evolution of services for finding current best evidence. Evidence-Based Medicine 4 (2), 37–38. DOI: 10.1136/ebmh.4.2.37 (cit. on pp. 161, 166).

Haynes, R. B. 2006. Of studies, syntheses, synopses, summaries, and systems. The "5S" evolution of information services for evidence-based healthcare decisions. Evidence-Based Medicine 11 (6), 162–164. DOI: 10.1136/ebm.11.6.162-a (cit. on pp. 161, 166).

Hazlett, A. 2015. The maturation of the Gettier problem. Philosophical Studies 172 (1), 1–6. DOI: 10.1007/s11098-014-0385-x (cit. on p. 52).

Heijenoort, J. van, ed. From Frege to Gödel. A Source Book in Mathematical Logic, 1879–1931 1967. Cambridge, MA: Harvard University Press.

Hempel, C. G. 1945. Studies in the Logic of Confirmation. Mind 54 (213), 1–23. ⟨URL: http://www.jstor.org/stable/2250886⟩ (cit. on p. 79).

Henkin, L. 1949. The Completeness of the First-Order Functional Calculus. The Journal of Symbolic Logic 14 (3), 159–166. DOI: 10.2307/2267044. ⟨URL: http://www.jstor.org/stable/2267044⟩ (cit. on p. 60).

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 p. 131).

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 p. 131).

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 p. 131).

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 p. 131).

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 p. 131).

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. 85, 86).

Hilbert, D. and Ackermann, W. 1928. Grundzüge der Theoretischen Logik. Die Grundlehren der Matematischen Wissenschaften XXVII. Berlin: Springer (cit. on p. 37).

Hilbert, D. 1967. On the infinite. In From Frege to Gödel. A Source Book in Mathematical Logic, 1879–1931. Ed. by Heijenoort, J. van. Trans. by Bauer-Mengelberg, S. Cambridge, MA: Harvard University Press, 367–392 (cit. on p. 51).

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. 38).

Hitt, J. 2001. The Year in Ideas: A to Z. Evidence-Based Medicine. New York Times. ⟨URL: http://www.nytimes.com/2001/12/09/magazine/the-year-in-ideas-a-to-z-evidence-based-medicine.html⟩ [visited on 2015-04-21] (cit. on p. 16).

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 p. 131).

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. 35).

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. 35).

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. 39).

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. 41, 42, 131, 147).

Hoc, J.-M., Green, T. R. G., Samurçay, R., and Gilmore, D. J., eds. Psychology of Programming 1990. London: Academic (cit. on p. 41).

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. 131, 136, 137).

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. 131, 136, 137).

Hoenig, J. M. and Heisey, D. M. 2001. The Abuse of Power. The Pervasive Fallacy of Power Calculations in Data Analysis. American Statistician 55 (1), 19–24. DOI: 10.1198/000313001300339897 (cit. on p. 73).

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. 131, 140).

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. 40).

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. 22).

HOPL III. Proceedings of the third ACM SIGPLAN conference on History of programming languages 2007. New York: ACM (cit. on p. 40).

Höst, M. and Runeson, P. 2007. Checklists for software engineering case study research. In ESEM 2007. Proceedings of the First International Symposium on Empirical Software Engineering and Measurement, 245–254. DOI: 10.1109/ESEM.2007.46 (cit. on p. 167).

House of Commons Science and Technology Committee 2010. Evidence Check 2. Homeopathy. ⟨URL: http://www.publications.parliament.uk/pa/cm200910/cmselect/cmsctech/45/45.pdf⟩ [visited on 2015-11-06] (cit. on p. 161).

Howick, J. 2011. The Philosophy of Evidence-Based Medicine. Oxford: Blackwell & BMJ Books. ⟨URL: https://play.google.com/store/books/details?id=O8djbHBva5IC⟩ (cit. on pp. 153, 160).

Howson, C. and Urbach, P. 2006. Scientific Reasoning. The Bayesian Approach. 3rd ed. Chicago: Open Court (cit. on pp. 51, 58, 69, 70).

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 p. 131).

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 p. 131).

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. 41).

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 p. 131).

Hughes, J. 1989. Why Functional Programming Matters. The Computer Journal 32 (2), 98–107. DOI: 10.1093/comjnl/32.2.98 (cit. on p. 28).

Hume, D. 2003. A Treatise of Human Nature. Mineola, NY: Dover. ⟨URL: https://play.google.com/store/books/details?id=_LPCAgAAQBAJ⟩ (cit. on pp. 56, 80).

Hume, D. 2011. An Enquiry Concerning Human Understanding. Ed. by Selby-Bigge, L. A. Project Gutenberg. ⟨URL: http://www.gutenberg.org/ebooks/9662⟩ (cit. on pp. 16, 57).

Hyland, K. 2001a. Bringing in the Reader. Addressee Features in Academic Articles. Written Communication 18 (4), 549–574. DOI: 10.1177/0741088301018004005 (cit. on p. 19).

Hyland, K. 2001b. Humble servants of the discipline? Self-mention in research articles. English for Specific Purposes 20 (3), 207–226. DOI: 10.1016/S0889-4906(00)00012-0 (cit. on pp. 18, 53).

IEEE Standard Glossary of Software Engineering Terminology 1990. IEEE Std 610.12-1990. DOI: 10.1109/IEEESTD.1990.101064 (cit. on pp. 21, 26).

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. 22).

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. 84).

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 p. 34).

Information Technology – Programming Languages – C 2011. INCITS/ISO/IEC 9899 (cit. on p. 34).

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 p. 74).

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. 37).

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. 125, 131, 134, 135, 172).

Ivarsson, M. and Gorschek, T. 2011. A Method for Evaluating Rigor and Industrial Relevance of Technology Evaluations. Empirical Software Engineering 16 (3), 365–395. DOI: 10.1007/s10664-010-9146-4 (cit. on p. 167).

Iverson, K. E. 1962. A Programming Language. New York: Wiley (cit. on p. 104).

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. 88).

Jammer, M. 1999. Concepts of Force. A Study in the Foundations of Dynamics. Mineola, NY: Dover. ⟨URL: https://play.google.com/store/books/details?id=LVe8AQAAQBAJ⟩ (cit. on p. 48).

Jaynes, E. T. 2003. Probability Theory. The Logic of Science. Ed. by Bretthorst, G. L. Cambridge University Press. ⟨URL: http://amzn.com/B00AKE1Q40⟩ (cit. on pp. 58, 59, 64).

Jeffrey, R. 2004. Subjective Probability. The Real Thing. Cambridge University Press (cit. on pp. 68, 70).

Jeffreys, H. 1931. Scientific Inference. Cambridge University Press. ⟨URL: https://archive.org/details/scientificinfere029500mbp⟩ [visited on 2015-03-29] (cit. on p. 58).

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 p. 131).

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 p. 131).

Johnstone Jr., H. W. 1963. Some Reflections on Argumentation. Logique & Analyse 6 (21–24), 30–39. ⟨URL: http://virthost.vub.ac.be/lnaweb/ojs/index.php/LogiqueEtAnalyse/article/view/180⟩ [visited on 2015-07-20] (cit. on p. 54).

Joyce, J. M. 1998. A Nonpragmatic Vindication of Probabilism. Philosophy of science 65 (4), 575–603. ⟨URL: http://www.jstor.org/stable/188574⟩ (cit. on p. 68).

Joyce, J. M. 2005. How Probabilities Reflect Evidence. Philosophical Perspectives 19: Epistemology, 153–178. DOI: 10.1111/j.1520-8583.2005.00058.x (cit. on pp. 76, 78).

Joyce, J. M. 2010. A Defense of Imprecise Credences in Inference and Decision-Making. Philosophical Perspectives 24: Epistemology, 281–323. DOI: 10.1111/j.1520-8583.2010.00194.x (cit. on p. 76).

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. 22).

Kaijanaho, A.-J. 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. University of Jyväskylä. ⟨URL: http://urn.fi/URN:ISBN:978-951-39-5791-9⟩ (cit. on pp. 5, 17, 21, 39, 78, 82, 109, 111, 123–125, 127, 128, 132, 148, 162, 166).

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. 37).

Karp, C. R. 1965. Finite-Quantifier Equivalence. In The Theory of Models. Proceedings of the 1963 international symposium at Berkeley. Ed. by Addison, J. W., Henkin, L., and Tarski, A. Studies in logic and the foundations of mathematics. Amsterdam: North-Holland (cit. on pp. 59, 66, 70).

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. 28).

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. 33).

Kell, S. 2014. In Search of Types. In Onward! 2014. Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software. DOI: 10.1145/2661136.2661154 (cit. on p. 39).

Kemeny, J. G. 1955. Fair Bets and Inductive Probabilities. Journal of Symbolic Logic 20 (3), 263–273. ⟨URL: http://www.jstor.org/stable/2268222⟩ (cit. on p. 68).

Kernighan, B. W. and Ritchie, D. M. 1978. The C programming language. Prentice-Hall (cit. on p. 34).

Kernighan, B. W. and Ritchie, D. M. 1988. The C programming language. 2nd ed. Prentice Hall (cit. on p. 34).

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. 131, 134, 135).

Keynes, J. M. 2014. A Treatise on Probability. Project Gutenberg. ⟨URL: http://www.gutenberg.org/ebooks/32625⟩ (cit. on p. 58).

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. 28).

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. 28).

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. 148).

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. 84).

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. 19).

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. 84).

Kitchenham, B. A., Brereton, O. P., Budgen, D., and Li, Z. 2009. An Evaluation of Quality Checklist Proposals. A participant-observer case study. In EASE'09. ⟨URL: http://www.bcs.org/upload/pdf/ewic_ea09_s3paper1.pdf⟩ (cit. on p. 167).

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. 88).

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. 83, 85, 86, 88).

Kitchenham, B. A., Burn, A. J., and Li, Z. 2009. A Quality Checklist for Technology-Centered Testing Studies. In EASE'09. ⟨URL: http://www.bcs.org/upload/pdf/ewic_ea09_s5paper3.pdf⟩ (cit. on p. 167).

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 (cit. on pp. 16, 156, 157).

Kitchenham, B. A., Sjøberg, D. I. K., Dybå, T., Pfahl, D., Brereton, P., Budgen, D., Höst, M., and Runeson, P. 2012. Three Empirical Studies on the Agreement of Reviewers about the Quality of Software Engineering Experiments. Information and Software Technology 54 (8), 804–819. DOI: 10.1016/j.infsof.2011.11.008 (cit. on p. 167).

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. 83).

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. 84–86, 88–90, 149).

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. 83, 88, 91).

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. 91).

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. 17, 83–86, 88–91, 93, 124, 218).

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. 83).

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. 131, 134, 135).

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. 131, 134, 135).

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 p. 131).

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 p. 131).

Klerer, M. 1991. Design of Very High-Level Computer Languages. A User-Oriented Approach. 2nd ed. 1991: McGraw-Hill (cit. on p. 42).

Knuth, D. E. 1992. Two Notes on Notation. American Mathematical Monthly 99 (5), 403–422. DOI: 10.2307/2325085 (cit. on p. 104).

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. 35, 40).

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. 24).

Kolmogorov, A. N. 1956. Foundations of the Theory of Probability. Trans. by Morrison, N. Second English Edition. New York: Chelsea. ⟨URL: http://www.mathematik.com/Kolmogorov/index.html⟩ (cit. on p. 80).

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 p. 131).

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. 25).

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. 31).

Kraemer, H. C. 1979. Ramifications of a population model for κ as a coefficient of reliability. Psychometrika 44 (4), 461–472 (cit. on p. 96).

Kripke, S. A. 1972. Naming and Necessity. In Semantics of Natural Language. Ed. by Davidson, D. and Harman, G. 2nd ed. Dordrecht, Holland: Reidel. DOI: 10.1007/978-94-010-2557-7_9 (cit. on p. 60).

Krippendorff, K. 1970. Bivariate Agreement Coefficients for Reliability of Data. In Sosiological Methodology 1970. Ed. by Borgatta, E. F., Bohrnstedt, G. W., et al. San Francisco: Jossey-Bass. Chap. 8, 139–150. ⟨URL: http://www.jstor.org/stable/270787⟩ (cit. on pp. 98, 102, 103).

Krippendorff, K. 1992. Recent Developments in Reliability Analysis. In 42nd Annual Meeting of the International Communication Association in Miami, Florida, May 21-25, 1992. ⟨URL: http://repository.upenn.edu/asc_papers/44/⟩ [visited on 2014-10-30] (cit. on p. 106).

Krippendorff, K. 2004. Reliability in Content Analysis. Some Common Misconceptions and Recommendations. Human Communications Research 30 (3), 411–433. DOI: 10.1111/j.1468-2958.2004.tb00738.x (cit. on pp. 17, 98, 99, 106, 107).

Krippendorff, K. 2011. Computing Krippendorff's Alpha-Reliability. ⟨URL: http://web.asc.upenn.edu/usr/krippendorff/mwebreliability5.pdf⟩ [visited on 2014-10-29] (cit. on pp. 96, 98, 99, 102–104, 106).

Krippendorff, K. 2013. Content Analysis. An Introduction to Its Methodology. 3rd ed. Los Angeles: SAGE (cit. on pp. 93, 96, 98, 99, 102–104, 106, 107).

Krippendorff, K. and Fleiss, J. L. 1978. Reliability of Binary Attribute Data. Correspondence. Biometrics 34 (1), 142–144. DOI: http://www.jstor.org/stable/2529602 (cit. on pp. 95, 107).

Krishnamurthi, S. 2008. Teaching Programming Languages in a Post-Linnaean Age. ACM SIGPLAN Notices 43 (11). DOI: 10.1145/1480828.1480846 (cit. on p. 29).

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. 35).

Kuhn, T. S. 1996. The Structure of Scientific Revolutions. 3rd ed. University of Chicago Press (cit. on pp. 18, 27, 146, 152).

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 p. 131).

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. 36).

Lakatos, I. 1976. Proofs and Refutations. The Logic of Mathematical Discovery. Ed. by Worrall, J. and Zahar, E. Cambridge University Press. ⟨URL: http://amzn.com/B00D2WQFA2⟩ (cit. on pp. 18, 52, 53, 55).

Lakatos, I. 1978. Falsification and the Methodology of Scientific Research Programmes. In Philosophical Papers. Vol. 1: The methodology of scientific research programmes. Ed. by Worrall, J. and Currie, G. Cambridge University Press, 8–101 (cit. on pp. 18, 186).

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. 38).

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 pp. 107, 122).

Langford, C. H. 1968. The Notion of Analysis in Moore's Philosophy. In The Philosophy of G. E. Moore. Ed. by Schlipp, P. A. 3rd ed. La Salle, IL: Open Court. Chap. 12, 321–342 (cit. on pp. 46, 47).

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 p. 131).

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. 22).

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. 28).

Leblanc, H. and Roeper, P. 1989. On Relativizing Kolmogorov's Absolute Probability Functions. In Memory of Andrei N. Kolmogorov (1903–1987). Notre Dame Journal of Formal Logic 30 (485–512). DOI: 10.1305/ndjfl/1093635234 (cit. on p. 61).

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. 131, 140).

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. 131, 139, 140).

Lee, P. M. 2012. Bayesian Statistics. An Introduction. 4th ed. Wiley. ⟨URL: https://play.google.com/store/books/details?id=WOW0KqWQwAcC⟩ (cit. on pp. 72, 79).

Lehman, R. S. 1955. On Confirmation and Rational Betting. Journal of Symbolic Logic 20 (3), 251–262. ⟨URL: http://www.jstor.org/stable/2268221⟩ (cit. on p. 68).

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. 88).

Lewis, D. 1979. Attitudes *de dicto* and *de se*. The Philosophical Review 88 (4), 513–543. DOI: 10.2307/2184843 (cit. on p. 60).

Lewis, D. 1980. A Subjectivist's Guide to Objective Chance. In Studies in Inductive Logic and Probability. Ed. by Jeffrey, R. Vol. II. Berkeley: University of California Press, 263–293 (cit. on p. 69).

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 p. 131).

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 p. 131).

Lieberman, H. 1986. Using Prototypical Objects to Implement Shared Behavior in Object Oriented Systems. In OOPSLA '86. Conference proceedings on Object-oriented programming systems, languages and applications, 214–223. DOI: 10.1145/28697.28718 (cit. on p. 56).

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 p. 131).

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. 26).

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. 84).

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. 37).

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. 131, 140).

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. 29).

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. 131, 134, 135).

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. 131, 136, 137, 177–180, 183).

Luff, M. 2013. Communication for Programmability and Performance on Multi-Core Processors. Tech. rep. UCAML-CL-TR-831. University of Cambridge Computer Laboratory. ⟨URL: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-831.html⟩ (cit. on pp. 177–180, 183).

Mac Lane, S. 1986. Mathematics. Form and Function. New York: Springer. DOI: 10.1007/978-1-4612-4872-9 (cit. on p. 51).

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. 83).

MacKenzie, D. 2005. Computing and the cultures of proving. Philosophical Transactions of the Royal Society. A: Mathematical, Physical & Engineering Sciences 363 (1835), 2335–2350. DOI: 10.1098/rsta.2005.1649 (cit. on p. 51).

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 p. 131).

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. 40).

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. 117, 131, 139, 140, 144).

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. 89).

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. 40).

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 p. 42).

Marlow, S., ed. Haskell 2010 Language Report 2010. ⟨URL: http://www.haskell.org/onlinereport/haskell2010⟩ (cit. on p. 23).

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. 85).

Martin, J. 1982. Application Development Without Programmers. Englewood Cliffs: Prentice-Hall (cit. on p. 26).

Martin, J. 1985. Fourth-Generation Languages. Vol. I: Principles. Englewood Cliffs: Prentice–Hall (cit. on p. 26).

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. 131, 134, 135).

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. 42, 131, 134, 135).

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 p. 131).

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. 30).

McCarthy, J. 1959. Letter to the editor. Communications of the ACM 2 (8), 2–3. DOI: perlis58:_prelim_repor (cit. on p. 32).

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. 36, 38).

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. 35).

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 p. 131).

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 p. 131).

McKee, M. 2013. First proof that infinitely many prime numbers come in pairs. Nature. DOI: 10.1038/nature.2013.12989 (cit. on p. 77).

Meacham, C. J. G. 2010. Two Mistakes Regarding the Principal Principle. British Journal for the Philosophy of Science 61 (2), 407–431. DOI: 10.1093/bjps/axp044 (cit. on p. 69).

Meehl, P. E. and Rosen, A. 1955. Antecedent Probability and the Efficiency of Psychometric Signs, Patterns, or Cutting Scores. Psychological Bulletin 52 (3), 194–216 (cit. on p. 74).

Meek, B. 1990. The static semantics file. SIGPLAN Notices 25 (4), 33–42. DOI: 10.1145/987481.987483 (cit. on p. 31).

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. 41).

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. 131, 136, 137).

Mielke Jr., P. W., Berry, K. J., and Johnston, J. E. 2008. Resampling Probability Values for Weighted Kappa with Multiple Raters. Psychological Reports 102 (2), 606–613. DOI: 10.2466/pr0.102.2.606-613 (cit. on pp. 103, 104).

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. 36).

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. 131, 140).

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. 131, 140).

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. 38).

Ministry of Education Publication Type Classification 2010. Ministry of Education and Culture of Finland. ⟨URL: https://aaltodoc.aalto.fi/doc_public/ohjeet/publicationclassification2010.pdf⟩ [visited on 2015-06-19] (cit. on p. 91).

Mitchell, J. C. 1993. On abstraction and the expressive power of programming languages. Science of Computer Programming 21 (2), 141–163. DOI: 10.1016/0167-6423(93)90004-9 (cit. on p. 15).

Mizrahi, M. 2012. Intuition Mongering. The Reasoner 6 (11), 169–170. ⟨URL: http://www.kent.ac.uk/secl/philosophy/jw/TheReasoner/vol6/TheReasoner-6(11).pdf⟩ (cit. on p. 53).

Mizrahi, M. 2013. More Intuition Mongering. The Reasoner 7 (1), 5–6. ⟨URL: http://www.kent.ac.uk/secl/philosophy/jw/TheReasoner/vol7/TheReasoner-7(1).pdf⟩ (cit. on p. 53).

Mizrahi, M. 2014. Does the Method of Cases Rest on a Mistake? Review of Philosophy and Psychology 5 (2), 183–197. DOI: 10.1007/s13164-013-0164-1 (cit. on p. 52).

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. 38).

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 p. 131).

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. 35).

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. 31).

Motulsky, H. J. 2015. Common misconceptions about data analysis and statistics. British Journal of Pharmacology 172 (8), 2126–2132. DOI: 10.1111/bph.12884 (cit. on p. 169).

Muschevichi, R., Potanin, A., Tempero, E., and Noble, J. 2008. Multiple Dispatch in Practice. In OOPSLA 2008 Conference Proceedings. 23rd Annual ACM Conference on Object-Oriented Programming, Systems, Languages, and Applications. DOI: 10.1145/1449764.1449808 (cit. on p. 162).

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 p. 131).

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. 42).

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. 42, 110).

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 p. 131).

Nagel, J. 2012. Intuitions and Experiments. A Defense of the Case Method in Epistemology. Philosophy and Phenomenological Research 85 (3), 495–527. DOI: 10.1111/j.1933-1592.2012.00634.x (cit. on p. 52).

Nagel, J., San Juan, V., and Mar, R. A. 2013. Lay denial of knowledge for justified true beliefs. Cognition 129 (3), 652–661. DOI: 10.1016/j.cognition.2013.02.008 (cit. on p. 53).

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. 131, 134, 135).

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. 131, 134, 135).

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. 131, 134, 135).

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. 41).

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. 31, 32, 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. 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. 32, 35).

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 p. 131).

Nelson, T. 1999. Ted Nelson's Computer Paradigm, Expressed as One-Liners. ⟨URL: http://xanadu.com.au/ted/TN/WRITINGS/TCOMPARADIGM/tedCompOneLiners.html⟩ [visited on 2015-03-16] (cit. on p. 50).

Newton, I. 2012. Philosophiae Naturalis Principia Mathematica. Trans. and annot. by Bruce, I. 3rd ed. Ian Bruce. ⟨URL: http://17centurymaths.com/contents/newtoncontents.html⟩ [visited on 2015-01-09] (cit. on p. 47).

Neyman, J. 1942. Basic Ideas and Some Recent Results of the Theory of Testing Statistical Hypotheses. Journal of the Royal Statistical Society 105 (4), 292–327. ⟨URL: http://www.jstor.org/stable/2980436⟩ (cit. on p. 73).

Neyman, J. and Pearson, E. S. 1933. On the Problem of the most Efficient Tests of Statistical Hypotheses. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 231, 289–337. ⟨URL: http://www.jstor.org/stable/91247⟩ (cit. on p. 72).

Neyman, J. and Pearson, E. S. 1967. Contributions to the Theory of Testing Statistical Hypotheses. Part III. In Joint Statistical Papers. Cambridge, England: University Press, 276–297 (cit. on p. 73).

Nichols, H. 1891. The Psychology of Time. American Journal of Psychology 3 (4), 453–529. DOI: 10.2307/1412061 (cit. on p. 84).

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. 91).

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. 40, 41).

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. 131, 134, 135).

Norris, C. 2011. Hawking contra Philosophy. Philosophy Now 82. ⟨URL: https://philosophynow.org/issues/82/Hawking_contra_Philosophy⟩ [visited on 2015-06-19] (cit. on p. 46).

Norton, J. 1991. Thought Experiments in Einstein's Work. In Thought Experiments in Science and Philosophy. Ed. by Horowitz, T. and Massey, G. J. Savage, MD: Rowman & Littlefield. ⟨URL: http://philsci-archive.pitt.edu/id/eprint/3190⟩ (cit. on p. 52).

Norton, J. D. 2010. There Are No Universal Rules of Induction. Philosophy of Science 77 (5), 765–777. DOI: 10.1086/656542 (cit. on p. 51).

Nutley, S., Morton, S., Jung, T., and Boaz, A. 2010. Evidence and policy in six European countries. Diverse approaches and common challenges. Evidence & Policy 6 (2), 131–144. DOI: 10.1332/174426410X502275 (cit. on pp. 16, 156).

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 p. 131).

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. 131, 140).

Olsen, N. S. 2014. Philosophical Theory-Construction and the Self-Image of Philosophy. Open Journal of Philosophy 4 (3), 231–243. DOI: 10.4236/ojpp.2014.43031 (cit. on p. 54).

OOPSLA '86. Conference proceedings on Object-oriented programming systems, languages and applications 1986.

O'Regan, G. 2012. A Brief History of Computing. 2nd ed. London: Springer. DOI: 10.1007/978-1-4471-2359-0 (cit. on p. 26).

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. 83).

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. 29).

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. 23, 24).

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. 42).

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. 42, 131).

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. 42, 131).

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. 42).

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. 126, 131).

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. 131, 134, 135, 177–180, 183).

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://nbn-resolving.org/urn:nbn:de:swb:90-129802⟩ (cit. on pp. 131, 134, 135, 177–180, 183).

paradigm, n. 2014. In OED Online. Mar. 2014. ⟨URL: http://www.oed.com/view/Entry/137329⟩ [visited on 2014-04-10] (cit. on p. 27).

Paseau, A. C. 2015. What's the point of complete rigour? Mind. ⟨URL: http://www.philosophy.ox.ac.uk/__data/assets/pdf_file/0004/36652/PaseauWebsiteMind2015.pdf⟩ [visited on 2015-07-20]. Forthcoming (cit. on p. 57).

Passmore, J. 1970. Philosophical Reasoning. 2nd ed. London: Duckworth (cit. on p. 52).

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 p. 131).

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. 19).

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 p. 131).

Pearson, K. 1901. Mathematical Contributions to the Theory of Evolution.—IX. On the Principle of Homotyposis and its Relation to Heredity, to the Variability of the Individual, and to that of the Race. Part I.—Homotyposis in the Vegetable Kingdom. Philosophical Transactions of the Royal Society of London. Series A, Containing Papers of a Mathematical or Physical Character 197, 285–379. DOI: 10.1098/rsta.1901.0020 (cit. on p. 98).

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. 83).

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. 32, 35).

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. 41).

perlsyn. Perl syntax 2014. Version 5.18.2. ⟨URL: http://perldoc.perl.org/perlsyn.pdf⟩ [visited on 2014-04-16] (cit. on pp. 33, 34).

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 p. 131).

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. 89).

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. 84–86, 88, 90).

Petersen, K., Vakkalank, S., and Kuzniarz, L. 2015. Guidelines for Conducting Systematic Mapping Studies in Software Engineering. Information and Software Technology 64, 1–18. DOI: 10.1016/j.infsof.2015.03.007 (cit. on pp. 85, 89, 91, 92).

Petticrew, M. and Roberts, H. 2006. Systematic Reviews in the Social Sciences. A Practical Guide. Malden, MA: Blackwell (cit. on pp. 83, 85–89).

Pfeifer, N. 2013. On Argument Strength. In Bayesian Argumentation. The practical side of probability. Ed. by Zenker, F. Synthese Library: Studies in Epistemology, Logic, Methodology, and Philosophy of Science 362. Dordrecht: Springer, 185–193. DOI: 10.1007/978-94-007-5357-0_10 (cit. on p. 59).

Pierce, B. C. 2002. Types and Programming Languages. Cambridge, Massachusetts: MIT Press (cit. on pp. 30, 37, 38).

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. 83).

Plant, B. 2012. Philosophical Diversity and Disagreement. Metaphilosophy 43 (5), 567–591. DOI: 10.1111/j.1467-9973.2012.01770.x (cit. on pp. 45, 54).

Plato 1984. The Being of the Beautiful. Plato's Theaetetus, Sophist, and Statesman. Trans. by Benardete, S. Chicago: University of Chicago Press (cit. on p. 46).

Plato 2005. Meno and other dialogues. Charmides, Laches, Lysis, Meno. Trans. by Waterfield, R. Oxford World's Classics. Oxford: Oxford University Press (cit. on p. 46).

Plato 2014. Socrates' Defense. Trans. by Woods, C. and Pack, R. ⟨URL: http://ssrn.com/abstract=1023144⟩ [visited on 2014-08-14] (cit. on p. iii).

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 p. 131).

Pollard, D. 2002. A User's Guide to Measure Theoretic Probability. Cambridge University Press. ⟨URL: https://play.google.com/store/books/details?id=t_8LBAAAQBAJ⟩ (cit. on p. 80).

Popper, K. 2002. The Logic of Scientific Discovery. London: Routledge. ⟨URL: http://amzn.com/B000OT7WLC⟩ (cit. on pp. 18, 80).

Popper, K. R. 1959. The Propensity Interpretation of Probability. British Journal for the Philosophy of Science 10 (37), 25–42. ⟨URL: http://www.jstor.org/stable/685773⟩ (cit. on p. 69).

Prechelt, L. 2000. An empirical comparison of seven programming languages. Computer 33 (10), 23–29. DOI: 10.1109/2.876288 (cit. on pp. 131, 144).

Prechelt, L. and Tichy, W. F. 1996. An experiment to assess the benefits of inter-module type checking. In Software Metrics Symposium, 1996., Proceedings of the 3rd International, 112–119. DOI: 10.1109/METRIC.1996.492448 (cit. on pp. 131, 134, 135, 144).

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. 117, 131, 134, 135, 144).

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. 131, 144).

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. 131, 134, 135).

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. 27).

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. 25).

programming, n. 2013. In OED Online. Dec. 2013. ⟨URL: http://www.oed.com/view/Entry/152232⟩ [visited on 2014-02-03] (cit. on p. 21).

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 p. 131).

Pullin, A. S. and Knight, T. M. 2001. Effectiveness in Conservation Practice. Pointers from Medicine and Public Health. Conservation Biology 15 (1), 50–54. DOI: 10.1111/j.1523-1739.2001.99499.x (cit. on pp. 16, 156).

Pullin, A. S. and Knight, T. M. 2003. Support for decision-making in conservation practice. An evidence-based approach. Journal for Nature Conservation 11 (2), 83–90. DOI: 10.1078/1617-1381-00040 (cit. on p. 156).

Pullum, G. K. 2014. Fear and loathing of the English passive. Language & Communication 37, 60–74. DOI: 10.1016/j.langcom.2013.08.009 (cit. on p. 19).

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. 131, 140).

Quine, W. V. 1968. Propositional Objects. Crítica. Revista Hispanoamericana de Filosofía 2 (5), 3–29. ⟨URL: http://critica.filosoficas.unam.mx/pg/en/numeros_detalle_articulo.php?id_articulo=980&id_volumen=97⟩ (cit. on p. 60).

Quine, W. V. 1993. In praise of observation sentences. Journal of Philosophy 90 (3), 107–116. ⟨URL: http://www.jstor.org/stable/2940954⟩ (cit. on p. 76).

Quine, W. V. O. 1960. Word and Object. MIT Press (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. 35).

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 p. 131).

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. 114).

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. 37).

Ramsey, F. P. 1999. Truth and Probability. Originally published posthumously in "The Foundations of Mathematics and other Logical Essays". ⟨URL: http://socserv.mcmaster.ca/econ/ugcm/3ll3/ramseyfp/ramsess.pdf⟩ [visited on 2015-04-05] (cit. on p. 68).

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. 26).

Rayside, D. and Campbell, G. T. 2000. An Aristotelian Understanding of Object-Oriented Programming. In Proceedings of the 15th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, 337–353. DOI: 10.1145/353171.353194 (cit. on p. 56).

retroactive, adj. 2014. In OED Online. Dec. 2014. ⟨URL: http://www.oed.com/view/Entry/164480⟩ [visited on 2015-03-16] (cit. on p. 50).

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. 38).

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. 38).

Reynolds, J. C. 1998. Theories of Programming Languages. Cambridge University Press (cit. on p. 22).

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 p. 34).

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 p. 131).

Rorty, R. M. 1992. Introduction. Metaphilosophical Difficulties of Linguistic Philosophy. In The linguistic turn. Essays in philosophical method. Ed. by Rorty, R. M. A reprint, with two retrospective essays; originally published in 1967. Chicago: University of Chicago Press, 1–39. ISBN: 0-226-72569-3 (cit. on pp. 46, 49).

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. 40).

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. 40).

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 pp. 152, 153, 159).

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. 35).

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. 131, 134, 135, 177–180, 183).

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. 131, 134, 135, 177–180, 183).

Runeson, P. and Höst, M. 2009. Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering 14 (2), 131–164. DOI: 10.1007/s10664-008-9102-8. [Visited on 2011-11-18] (cit. on p. 167).

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. 145, 167).

Russell, B. 1905. On denoting. Mind. New series 14 (56), 479–493. DOI: 10.1093/mind/XIV.4.479. ⟨URL: http://www.jstor.org/stable/2248381⟩ (cit. on p. 51).

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. 37).

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. 37).

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. 40, 42).

Ryle, G. 1954. Proofs in Philosophy. Revue Internationale de Philosophie 8 (27–28), 150–157. ⟨URL: http://www.jstor.org/stable/23936966⟩ (cit. on p. 52).

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 p. 131).

Sackett, D. L. et al. 1981. How to read clinical journals. I. Why to read them and how to start reading them critically. Canadian Medical Association Journal 214 (5), 555–558. ⟨URL: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC1705173/⟩ [visited on 2015-04-21] (cit. on p. 16).

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. 41).

Sackman, H. 1970. Man–Computer Problem Solving. Experimental Evaluation of Time-Sharing and Batch Processing. Princeton: Auerbach (cit. on p. 41).

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 p. 131).

Saha, S., Koley, M., Hossain, S. I., Mundle, M., Ghosh, S., Nag, G., Datta, A. K., and Rath, P. 2013. Individualized homeopathy versus placebo in essential hypertension. A double-blind randomized controlled trial. Indian Journal of Research in Homeopathy 7 (2), 62–71. DOI: 10.4103/0974-7168.116629 (cit. on p. 161).

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. 31).

Salleh, N. and Nordin, A. 2014. Trends and Perceptions of Evidence-based Software Engineering Research in Malaysia. In The 5th International Conference on Information and Communication Technology for the Muslim World (ICT4M), 1–6. DOI: 10.1109/ICT4M.2014.7020597 (cit. on p. 157).

Sammet, J. E. 1969. Programming Languages. History and Fundamentals. Englewood Cliffs: Prentice–Hall (cit. on pp. 22, 26, 28, 35, 40).

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. 40).

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. 35, 41).

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. 40).

Santos, R. E. S., Magalhães, C. V. C. de, and Silva, F. Q. B. da 2014. The use of systematic reviews in Evidence Based Software Engineering. A Systematic Mapping Study. In 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM 2014). DOI: 10.1145/2652524.2652553 (cit. on p. 157).

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. 83).

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. 131, 140).

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. 131, 140).

Schwandt, T. A. 2007. The SAGE Dictionary of Qualitative Inquiry. SAGE. DOI: 10.4135/9781412986281 (cit. on p. 91).

Scott, D. S. 1993. A Type-Theoretical Alternative to ISWIM, CUCH, OWHY. Theoretical Computer Science 121 (1–2), 411–440. DOI: 10.1016/0304-3975(93)90095-B (cit. on p. 38).

Scott, D. and Krauss, P. 1966. Assigning Probabilities to Logical Formulas. In Aspects of Inductive Logic. Ed. by Hintikka, J. and Suppes, P. Studies in Logic and the Foundations of Mathematics 43. North-Holland, 219–264. DOI: 10.1016/S0049-237X(08)71672-0 (cit. on pp. 65, 66, 71).

Scott, W. A. 1955. Reliability of Content Analysis. The Case of Nominal Scale Coding. Public Opinion Quarterly 19 (3), 321–325. ⟨URL: http://www.jstor.org/stable/2746450⟩ (cit. on p. 96).

Sehon, S. R. and Stanley, D. E. 2003. A philosophical analysis of the evidence-based medicine debate. BMC Health Services Research 3, a14. DOI: 10.1186/1472-6963-3-14 (cit. on p. 154).

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. 126, 131, 139, 140, 143).

Sellars, W. 1962. Philosophy and the Scientific Image of Man. In Frontiers of Science and Philosophy. Ed. by Colodny, R. G. University of Pittsburgh Series in the Philosophy of Science 1. University of Pittsburgh Press, 35–78. ⟨URL: http://digital.library.pitt.edu/cgi-bin/t/text/text-idx?idno=31735057893632;view=toc;c=pittpress⟩ (cit. on p. 51).

Sethi, R. 1996. Programming Languages. Concepts & Constructs. 2nd ed. Reading, MA: Addison–Wesley (cit. on p. 21).

Seyedsayamdost, H. 2015. On Normativity and Epistemic Intuitions. Failure of Replication. Episteme 12 (1), 95–116. DOI: 10.1017/epi.2014.27 (cit. on p. 53).

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. 84).

Shapiro, E. 1991. Separating Concurrent Languages with Categories of Language Embeddings. In STOC'91. Proceedings of the twenty-third annual ACM symposium on Theory of computing, 198–208. DOI: 10.1145/103418.103423 (cit. on p. 15).

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. 40).

Sheil, B. A. 1981. The Psychological Study of Programming. ACM Computing Surveys 13 (1), 101–120. DOI: 10.1145/356835.356840 (cit. on pp. 36, 41, 42, 131, 147).

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. In Proceedings of the AFIPS National Computer Conference, 1021. DOI: 10.1109/AFIPS.1979.59 (cit. on pp. 131, 134, 135).

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 pp. 16, 155).

Shimony, A. 1955. Coherence and the Axioms of Confirmation. Journal of Symbolic Logic 20 (1), 1–28. ⟨URL: http://www.jstor.org/stable/2268039⟩ (cit. on p. 68).

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 p. 131).

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. 131, 136, 137, 172).

Shneiderman, B. 1980. Software Psychology. Human Factors in Computer and Information Systems. Cambridge, MA: Winthrop (cit. on p. 41).

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. 35).

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. 131, 136, 137, 172).

Silva Neto, F. M. S. d. 2014. Rewriting Concurrent Haskell Programs to STM. MA thesis. Informatics Center of Federal University of Pernambuco. ⟨URL: http://repositorio.ufpe.br/handle/123456789/11435⟩ (cit. on pp. 177–180, 183).

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. 83).

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. 83).

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 p. 131).

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. 131, 136, 137, 143, 146, 172, 173, 183).

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. 33, 131, 136, 137, 143, 172, 173, 175, 176, 183).

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. 33, 131, 136, 137, 143, 172, 173, 183).

Simmonds, D. M. 2012. The Programming Paradigm Evolution. Computer 45 (6), 93–95 (cit. on p. 28).

Simmons, J. P., Nelson, L. D., and Simonsohn, U. 2011. False-Positive Psychology. Undisclosed Flexibility in Data Collection and Analysis Allows Presenting Anything as Significant. Psychological Science 22 (11), 1359–1366. DOI: 10.1177/0956797611417632 (cit. on p. 169).

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. 88).

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. 31).

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. 131, 138, 172).

Smith, P. 2003. An Introduction to Formal Logic. Cambridge University Press. ⟨URL: https://play.google.com/store/books/details?id=udELAQAAQBAJ⟩ (cit. on p. 59).

Smith, R. and Rennie, D. 2014. Evidence-Based Medicine. An Oral History. JAMA 311 (4), 365–367. DOI: 10.1001/jama.2013.286182 (cit. on p. 16).

Smithies, D. 2015. Ideal rationality and logical omniscience. Synthese. DOI: 10.1007/s11229-015-0735-z. prepublication (cit. on p. 69).

Snyder, A. 1986. Encapsulation and Inheritance in Object-Oriented Programming Languages. In OOPSLA '86. Conference proceedings on Object-oriented programming systems, languages and applications, 38–45. DOI: 10.1145/960112.28702 (cit. on p. 56).

Soare, R. I. 2007. Computability and Incomputability. In Computation and Logic in the Real World. Third Conference on Computability in Europe, CiE 2007, Siena, Italy, June 18–23, 2007, Proceedings. Ed. by Cooper, B., Löwe, B., and Sorbi, A. Lecture Notes in Computer Science 4497. Berlin, 705–715. DOI: 10.1007/978-3-540-73001-9_75 (cit. on p. 55).

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. 131, 136, 137).

Spinellis, D. 2005. Java Makes Scripting Languages Irrelevant? IEEE Software 22 (3), 70–71. DOI: 10.1109/MS.2005.67 (cit. on p. 29).

SpringerLink s.d. Search FAQ. ⟨URL: http://www.springerlink.com/help/search-tips.mpx⟩ [visited on 2011-10-19] (cit. on p. 116).

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. 39).

Steele Jr., G. L. 2006. Thoughts on Language Design. Dr. Dobb's Journal 31 (1), 31–32 (cit. on p. 39).

Stefik, A. 2014. The Elephant in the Room. In Future Directions in CS Education. ⟨URL: http://web.stanford.edu/~coopers/2013Summit/StefikAndreasUNLV.pdf⟩ (cit. on p. 162).

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. 126, 131, 139, 143, 144).

Stefik, A. and Hanenberg, S. 2014. The Programming Language Wars. Questions and Responsibilities for the Programming Language Community. In Onward! 2014. Proceedings of the 2014 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software. DOI: 10.1145/2661136.2661156 (cit. on p. 162).

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. In 22nd International Conference on Program Comprehension (ICPC 2014). Proceedings. DOI: 10.1145/2597008.2597154 (cit. on pp. 42, 147, 150, 162, 167).

Stefik, A. and Ladner, R. E. 2015. Introduction to AccessCS10K and Accessible Tools for Teaching Programming. In SIGCSE'15. Proceedings of the 46th ACM Technical Symposium on Computer Science Education. DOI: 10.1145/2676723.2677321 (cit. on p. 162).

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. 42, 162, 173, 174, 183).

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. 131, 162).

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. 40).

Stitch, S. P. 2001. Plato's Method Meets Cognitive Science. Free Inquiry 21 (2), 36–38. ⟨URL: http://www.rci.rutgers.edu/~stich/Publications/Papers/Plato_Method.pdf⟩ [visited on 2015-02-15] (cit. on p. 52).

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. 22).

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. 35).

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. 82, 86, 153, 154, 159, 161, 167–170).

Stroustrup, B. 1988. What is Object-Oriented Programming? IEEE Software 5 (3), 10–20. DOI: 10.1109/52.2020 (cit. on p. 28).

Stroustrup, B. 1994. The Design and Evolution of C++. Reading, Massachusetts: Addison-Wesley (cit. on p. 41).

Stroustrup, B. 2014. The C++ Programming Language. 4th ed. Upper Saddle River: Addison-Wesley (cit. on pp. 29, 40).

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. 131, 134, 135).

Sword, H. 2012. Stylish Academic Writing. Harvard University Press. ⟨URL: http://www.amazon.com/gp/product/B008J2GCZ8⟩ (cit. on pp. 18, 19).

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. 28).

Taivalsaari, A. 1996. On the Notion of Inheritance. ACM Computing Surveys 28 (3), 438–479. DOI: 10.1145/243439.243441 (cit. on pp. 28, 56).

Tarski, A. 1944. The Semantic Conception of Truth and the Foundations of Semantics. Philosophy and Phenomenological Research 4 (3), 341–376. ⟨URL: http://www.jstor.org/stable/2102968⟩ (cit. on p. 48).

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 p. 131).

Tedre, M. 2006. The Development of Computer Science. A Sociocultural Perspective. Computer Science Dissertations XIV. University of Joensuu. ⟨URL: http://urn.fi/URN:ISBN:952-458-867-6⟩ (cit. on p. 56).

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. 131, 136, 137).

The Python Language Reference 2014. ⟨URL: http://docs.python.org/3/reference/⟩ (cit. on p. 23).

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. 131, 140).

Thurston, W. P. 1994. On Proof and Progress in Mathematics. Bulletin of the American Mathematical Society. new 30 (2), 161–177. DOI: 10.1090/S0273-0979-1994-00502-6 (cit. on p. 51).

Tichy, W. F. 2000. Hints for Reviewing Empirical Work in Software Engineering. Empirical Software Engineering 5 (4), 309–312. DOI: 10.1023/A:1009844119158 (cit. on p. 146).

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 p. 131).

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. 145).

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. 89).

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 p. 131).

Toulmin, S. E. 2003. The Uses of Argument. Updated Edition. Cambridge University Press (cit. on p. 50).

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. 38).

Turing, A. M. 1937. On computable numbers, with an application to the Entscheidungsproblem. Proceedings of the London Mathematical Society s2–42 (1), 230–265. DOI: 10.1112/plms/s2-42.1.230 (cit. on pp. 54, 55).

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. 36).

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. 90).

Turner, R. 2007. Understanding Programming Languages. Minds & Machines 17 (2), 203–216. DOI: 10.1007/s11023-007-9062-6 (cit. on p. 22).

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. 22).

Turri, J. 2012. Is knowledge justified true belief? Synthese 184 (3), 247–259. DOI: 10.1007/s11229-010-9773-8 (cit. on p. 46).

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. 131, 134, 135).

UNIVAC FLOW-MATIC Programming System 1958. Remington–Rand Univac (cit. on p. 35).

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 p. 131).

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. 27, 29).

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 p. 149).

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. 126, 131, 136, 137, 172, 173, 175, 183).

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 p. 131).

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. 126, 131).

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. 38).

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 p. 131).

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 p. 131).

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 p. 131).

Walton, D. N. 1990. What is reasoning? What is an argument? Journal of Philosophy 87 (8), 399–419. ⟨URL: http://www.jstor.org/stable/2026735⟩ (cit. on p. 50).

Warrens, M. J. 2010. Inequalities between kappa and kappa-like statistics for $k \times k$ tables. Psychometrika 75 (1), 176–185. DOI: 10.1007/S11336-009-9138-8 (cit. on p. 99).

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. 88).

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. 40).

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. 28).

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. 27).

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. 131, 140).

Weinberg, G. M. 1971. The Psychology of Computer Programming. New York: Van Nostrand Reinhold (cit. on p. 41).

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. 28).

Wenmackers, S. and Romeijn, J.-W. 2015. New theory about old evidence. A framework for open-minded Bayesianism. Synthese. DOI: 10.1007/s11229-014-0632-x. prepublication (cit. on pp. 69, 71).

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. 131, 140, 144).

Wexelblat, R. L., ed. History of Programming Languages 1981. ACM monograph series. New York, NY: Academic (cit. on p. 40).

What Works Clearinghouse 2014. Procedures and Standards Handbook. Version 3.0. Institute of Education Sciences. ⟨URL: http://ies.ed.gov/ncee/wwc/pdf/reference_resources/wwc_procedures_v3_0_standards_handbook.pdf⟩ [visited on 2015-02-09] (cit. on p. 167).

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. 25).

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. 22).

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. 37).

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. 40).

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 p. 131).

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 p. 131).

Wieringa, R., Condori–Fernandez, N., Daneva, M., Mutschler, B., and Pastor, O. 2012. Lessons Learned from Evaluating a Checklist for Reporting Experimental and Observational Research. In ESEM'12. Proceedings of the ACM–IEEE International Symposium on Empirical Software Engineering and Measurement, 157–160. DOI: 10.1145/2372251.2372279 (cit. on p. 167).

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. 33).

Wilbanks, J. J. 2010. Defining Deduction, Induction, and Validity. Argumentation 24 (1), 107–124. DOI: 10.1007/s10503-009-9131-5 (cit. on p. 51).

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. 25).

Williamson, T. 2007. The Philosophy of Philosophy. The Blackwell/Brown lectures in philosophy 2. Malden, MA: Blackwell (cit. on pp. 47, 52).

Wirth, N. 1974. On the Design of Programming Languages. In IFIP Congress 74, 386–393 (cit. on p. 39).

Wirth, N. 1976. Algorithms + Data Structures = Programs. Englewood Cliffs: Prentice–Hall (cit. on p. 23).

Wittgenstein, L. 1974. Tractatus Logico-Philosophicus. Trans. by Pears, D. F. and McGuinness, B. F. With an intro. by Russell, B. London: Routledge (cit. on pp. 16, 19, 45, 46, 50, 53, 57).

Wittgenstein, L. 2009. Philosophical Investigations. Trans. by Anscombe, G. E. M., Hacker, P. M. S., and Schulte, J. 4th ed. Chichester: Wiley–Blackwell (cit. on p. 49).

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. 83, 88).

Worrall, J. 2010. For Universal Rules, Against Induction. Philosophy of Science 77 (5), 740–753. DOI: 10.1086/656823 (cit. on p. 51).

Wright, G. H. von 1957. The Logical Problem of Induction. 2nd ed. New York: Barnes & Noble. ⟨URL: https : / / archive . org / details / logicalproblemof010585mbp⟩ [visited on 2015-03-12] (cit. on p. 58).

Wyatt, A. 2002. Evidence Based Policy Making. The View From A Centre. Public Policy and Administration 17 (3), 12–28. DOI: 10.1177/095207670201700302 (cit. on p. 156).

Yin, R. K. 2009. Case Study Research. Design and Methods. 4th ed. Los Angeles: Sage (cit. on pp. 145, 178).

Zenker, F. 2013a. Bayesian Argumentation. The Practical Side of Probabaility. In Bayesian Argumentation. The practical side of probability. Ed. by Zenker, F. Synthese Library: Studies in Epistemology, Logic, Methodology, and Philosophy of Science 362. Dordrecht: Springer, 1–11. DOI: 10.1007/978-94-007-5357-0_1 (cit. on p. 59).

Zenker, F., ed. Bayesian Argumentation. The practical side of probability 2013b. Synthese Library: Studies in Epistemology, Logic, Methodology, and Philosophy of Science 362. Dordrecht: Springer.

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. 83).

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. 87, 92).

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. 87, 88, 92, 117, 148).

APPENDIX 1 PROOFS

Lemma 36.

$$P_e^{\alpha} = \frac{1}{2n_{..} - 1}\left(2n_{..}\sum_{i=1}^{m} {r_i}^2 - 1\right)$$

Proof. Let us calculate:

$$\begin{aligned}
P_e^{\alpha} &= \sum_{i=1}^{m}\left(\frac{n_{.i} + n_{i.}}{2n_{..}} \times \frac{n_{.i} + n_{i.} - 1}{2n_{..} - 1}\right) \\
&= \frac{1}{2n_{..}(2n_{..} - 1)}\sum_{i=1}^{m}(n_{.i} + n_{i.})(n_{.i} + n_{i.} - 1) \\
&= \frac{1}{2n_{..}(2n_{..} - 1)}\sum_{i=1}^{m}\left({n_{.i}}^2 + 2n_{i.}n_{.i} + {n_{i.}}^2 - n_{.i} - n_{i.}\right) \\
&= \frac{1}{2n_{..}(2n_{..} - 1)}\left(\sum_{i=1}^{m}(n_{.i} + n_{i.})^2 - 2n_{..}\right) \\
&= \frac{1}{2n_{..}(2n_{..} - 1)}\left(\sum_{i=1}^{m}4{n_{..}}^2{r_i}^2 - 2n_{..}\right) \\
&= \frac{1}{2n_{..} - 1}\left(2n_{..}\sum_{i=1}^{m}{r_i}^2 - 1\right)
\end{aligned}$$

□

Proof of Proposition 10 (p. 96). Let us calculate:

$$\begin{aligned}
P_e^{\kappa} &= \sum_{k=1}^{m} p_k q_k = \sum_{k=1}^{m}\left(\frac{1}{m} + \tilde{p}_k\right)\left(\frac{1}{m} + \tilde{q}_k\right) = \sum_{k=1}^{m}\left(\frac{1}{m^2} + \frac{1}{m}\tilde{p}_k + \frac{1}{m}\tilde{q}_k + \tilde{p}_k\tilde{q}_k\right) \\
&= \sum_{k=1}^{m}\frac{1}{m^2} + \frac{1}{m}\sum_{k=1}^{m}\tilde{p}_k + \frac{1}{m}\sum_{k=1}^{m}\tilde{q}_k + \sum_{k=1}^{m}\tilde{p}_k\tilde{q}_k = \frac{1}{m} + \sum_{k=1}^{m}\tilde{p}_k\tilde{q}_k. \\
P_e^{\pi} &= \sum_{k=1}^{m}{r_k}^2 = \sum_{k=1}^{m}\left(\frac{1}{m} + \tilde{r}_k\right)^2 = \sum_{k=1}^{m}\left(\frac{1}{m^2} + \frac{2}{m}\tilde{r}_k + \tilde{r}_k^2\right) = \frac{1}{m} + \sum_{k=1}^{m}\tilde{r}_k^2. \\
P_e^{\alpha} &= \frac{1}{2n_{..} - 1}\left(2n_{..}\sum_{k=1}^{m}{r_k}^2 - 1\right) = \frac{1}{2n_{..} - 1}\left(2n_{..}\left(\frac{1}{m} + \sum_{k=1}^{m}\tilde{r}_k^2\right) - 1\right) \\
&= \frac{1}{2n_{..} - 1}\left(2n_{..}\sum_{k=1}^{m}\tilde{r}_k^2 + \frac{2n_{..}}{m} - 1\right) = \frac{1}{2n_{..} - 1}\left(2n_{..}\sum_{k=1}^{m}\tilde{r}_k^2 + \frac{2n_{..} - m}{m}\right).
\end{aligned}$$

□

Proof of Proposition 12 (p. 98).

$$\begin{aligned}
P_e^{AC_1} &= \frac{1}{m-1}\sum_{k=1}^{m} r_k(1 - r_k) = \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{1}{m} + \tilde{r}_k\right)\left(1 - \left(\frac{1}{m} + \tilde{r}_k\right)\right) \\
&= \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{1}{m} + \tilde{r}_k\right)\left(\frac{m-1}{m} - \tilde{r}_k\right) \\
&= \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{1}{m}\frac{m-1}{m} - \frac{1}{m}\tilde{r}_k + \frac{m-1}{m}\tilde{r}_k - \tilde{r}_k^2\right)
\end{aligned}$$

$$= \frac{1}{m-1}\sum_{k=1}^{m}\left(\frac{m-1}{m^2}+\frac{m-2}{m}\tilde{r}_k-\tilde{r}_k^2\right)$$
$$= \frac{1}{m-1}\sum_{k=1}^{m}\frac{m-1}{m^2}+\frac{1}{m-1}\frac{m-2}{m}\sum_{k=1}^{m}\tilde{r}_k-\frac{1}{m-1}\sum_{k=1}^{m}\tilde{r}_k^2$$
$$= \frac{1}{m}-\frac{1}{m-1}\sum_{k=1}^{m}\tilde{r}_k^2. \qquad \square$$

Lemma 37.

$$D_o = \frac{1}{n_{..}}\sum_{i=1}^{m}\sum_{j=1}^{m} n_{ij}\delta_{ij} \qquad D_e^{\kappa} = \sum_{i=1}^{m}\sum_{j=1}^{m} p_i q_j \delta_{ij}$$
$$D_e^{\pi} = \frac{1}{4{n_{..}}^2}\sum_{i=1}^{m}(n_{.i}+n_{i.})\sum_{j=1}^{m}(n_{.j}+n_{j.})\delta_{ij}$$
$$D_e^{\alpha} = \frac{2n_{..}}{2n_{..}-1}D_e^{\pi} = \frac{1}{2n_{..}(2n_{..}-1)}\sum_{i=1}^{m}(n_{.i}+n_{i.})\sum_{j=1}^{m}(n_{.j}+n_{j.})\delta_{ij}$$

Proof.

$$D_o = 1-P_o = 1-\frac{1}{n_{..}}\sum_{i=1}^{m}n_{ii} = \frac{1}{n_{..}}\left(n_{..}-\sum_{i=1}^{m}n_{ii}\right) = \frac{1}{n_{..}}\sum_{i=1}^{m}(n_{i.}-n_{ii})$$
$$= \frac{1}{n_{..}}\sum_{i=1}^{m}\left(\sum_{j=1}^{m}n_{ij}-n_{ii}\right) = \frac{1}{n_{..}}\sum_{i=1}^{m}\sum_{j=1}^{m}n_{ij}\delta_{ij}.$$
$$D_e^{\kappa} = 1-P_e^{\kappa} = 1-\sum_{i=1}^{m}p_i q_i = 1-\sum_{i=1}^{m}\frac{n_{.i}}{n_{..}}\frac{n_{i.}}{n_{..}} = 1-\frac{1}{{n_{..}}^2}\sum_{i=1}^{m}n_{.i}n_{i.}$$
$$= 1-\frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\left(\sum_{j=1}^{m}n_{ji}\right)\left(\sum_{j=1}^{m}n_{ij}\right) = 1-\frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}n_{ji}n_{ik}$$
$$= 1-\frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}(1-\delta_{jk})$$
$$= 1-\frac{1}{{n_{..}}^2}\left(\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}-\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}\delta_{jk}\right)$$
$$= 1-\frac{1}{{n_{..}}^2}\left(\left(\sum_{i=1}^{m}\sum_{j=1}^{m}n_{ij}\right)\left(\sum_{i=1}^{m}\sum_{j=1}^{m}n_{ij}\right)-\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}\delta_{jk}\right)$$
$$= 1-\frac{1}{{n_{..}}^2}\left({n_{..}}^2-\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}\delta_{jk}\right)$$
$$= \frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}\delta_{jk} = \frac{1}{{n_{..}}^2}\sum_{j=1}^{m}\sum_{k=1}^{m}\left(\sum_{i=1}^{m}\sum_{\ell=1}^{m}n_{ij}n_{k\ell}\right)\delta_{jk}$$
$$= \frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{m}\sum_{\ell=1}^{m}n_{ki}n_{j\ell}\right)\delta_{ij} = \frac{1}{{n_{..}}^2}\sum_{i=1}^{m}\sum_{j=1}^{m}\left(\sum_{k=1}^{m}n_{ki}\right)\left(\sum_{\ell=1}^{m}n_{j\ell}\right)\delta_{ij}$$

$$= \frac{1}{{n_{..}}^2} \sum_{i=1}^{m} \sum_{j=1}^{m} n_{.i} n_{j.} \delta_{ij} = \sum_{i=1}^{m} \sum_{j=1}^{m} p_i q_j \delta_{ij}.$$

$$D_e^{\pi} = 1 - P_e^{\pi} = 1 - \sum_{i=1}^{m} {r_i}^2 = 1 - \sum_{i=1}^{m} \left(\frac{n_{.i} + n_{i.}}{2n_{..}} \right)^2 = 1 - \sum_{i=1}^{m} \frac{(n_{.i} + n_{i.})^2}{(2n_{..})^2}$$

$$= \frac{n_{..} + n_{..}}{2n_{..}} - \sum_{i=1}^{m} \frac{(n_{.i} + n_{i.})^2}{(2n_{..})^2} = \sum_{i=1}^{m} \frac{n_{.i} + n_{i.}}{2n_{..}} - \sum_{i=1}^{m} \frac{(n_{.i} + n_{i.})^2}{(2n_{..})^2}$$

$$= \sum_{i=1}^{m} \left(\frac{n_{.i} + n_{i.}}{2n_{..}} - \frac{(n_{.i} + n_{i.})^2}{(2n_{..})^2} \right) = \sum_{i=1}^{m} \frac{n_{.i} + n_{i.}}{2n_{..}} \left(1 - \frac{n_{.i} + n_{i.}}{2n_{..}} \right)$$

$$= \sum_{i=1}^{m} \frac{n_{.i} + n_{i.}}{2n_{..}} \left(\frac{2n_{..}}{2n_{..}} - \left(\frac{n_{.i}}{2n_{..}} + \frac{n_{i.}}{2n_{..}} \right) \right) = \sum_{i=1}^{m} \frac{n_{.i} + n_{i.}}{2n_{..}} \frac{1}{2n_{..}} \left(2n_{..} - (n_{.i} + n_{i.}) \right)$$

$$= \frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \left(2n_{..} - n_{.i} - n_{i.} \right)$$

$$= \frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \left(\sum_{j=1}^{m} n_{.j} - n_{.i} + \sum_{j=1}^{m} n_{j.} - n_{i.} \right)$$

$$= \frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \left(\sum_{j=1}^{m} n_{.j} \delta_{ij} + \sum_{j=1}^{m} n_{j.} \delta_{ij} \right)$$

$$= \frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \sum_{j=1}^{m} (n_{.j} + n_{j.}) \delta_{ij}.$$

$$D_e^{\alpha} = 1 - P_e^{\alpha} = 1 - \frac{1}{2n_{..} - 1} \left(2n_{..} \sum_{i=1}^{m} {r_i}^2 - 1 \right)$$

$$= 1 - \left(\frac{2n_{..}}{2n_{..} - 1} \sum_{i=1}^{m} {r_i}^2 - \frac{1}{2n_{..} - 1} \right)$$

$$= 1 + \frac{1}{2n_{..} - 1} - \frac{2n_{..}}{2n_{..} - 1} \sum_{i=1}^{m} {r_i}^2$$

$$= \frac{2n_{..}}{2n_{..} - 1} - \frac{2n_{..}}{2n_{..} - 1} \sum_{i=1}^{m} {r_i}^2$$

$$= \frac{2n_{..}}{2n_{..} - 1} \left(1 - \sum_{i=1}^{m} {r_i}^2 \right) = \frac{2n_{..}}{2n_{..} - 1} D_e^{\pi}.$$

$$D_e^{\alpha} = \frac{2n_{..}}{2n_{..} - 1} D_e^{\pi} = \frac{2n_{..}}{2n_{..} - 1} \left(\frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \sum_{j=1}^{m} (n_{.j} + n_{j.}) \delta_{ij} \right)$$

$$= \frac{1}{2n_{..}(2n_{..} - 1)} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \sum_{j=1}^{m} (n_{.j} + n_{j.}) \delta_{ij}. \qquad \square$$

Proposition 38.

$$D_o = \frac{1}{o_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}$$

$$D_e^{\alpha} \frac{1}{o_{..}(o_{..}-1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij} \qquad D_e^{\pi} = \frac{1}{{o_{..}}^2} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}$$

$$\alpha = 1 - (o_{..}-1) \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} \qquad \pi = 1 - o_{..} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}}$$

Proof. Let us calculate:

$$D_o = \frac{1}{n_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} n_{ij} \delta_{ij} = \frac{1}{2n_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} (n_{ij} + n_{ji}) \delta_{ij} = \frac{1}{o_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}.$$

$$D_e^{\alpha} = \frac{1}{2n_{..}(2n_{..}-1)} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \sum_{j=1}^{m} (n_{.j} + n_{j.}) \delta_{ij} = \frac{1}{o_{..}(o_{..}-1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}.$$

$$D_e^{\pi} = \frac{1}{4{n_{..}}^2} \sum_{i=1}^{m} (n_{.i} + n_{i.}) \sum_{j=1}^{m} (n_{.j} + n_{j.}) \delta_{ij} = \frac{1}{o_{\cdot\cdot}^2} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}.$$

$$\alpha = 1 - \frac{D_o}{D_e^{\alpha}} = 1 - \frac{\frac{1}{o_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\frac{1}{o_{..}(o_{..}-1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} = 1 - (o_{..}-1) \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}}.$$

$$\pi = 1 - \frac{D_o}{D_e^{\pi}} = 1 - \frac{\frac{1}{o_{..}} \sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\frac{1}{{o_{..}}^2} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} = 1 - o_{..} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}}.$$

□

Proof of Proposition 13 (p. 99). All the identities except the following are trivial:

$$D_e^{\alpha} = \frac{1}{o_{..}(o_{..}-1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij} = \frac{o_{..}}{{o_{..}}^2(o_{..}-1)} \sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij} = \frac{o_{..}}{o_{..}-1} \sum_{i=1}^{m} r_i \sum_{j=1}^{m} r_j \delta_{ij}.$$

□

Proof of Proposition 14 (p. 99). Let us calculate:

$$\begin{aligned}
\alpha - \pi &= \left(1 - (o_{..}-1) \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}}\right) - \left(1 - o_{..} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}}\right) \\
&= o_{..} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} - (o_{..}-1) \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} \\
&= (o_{..} - (o_{..}-1)) \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} \\
&= \frac{1}{o_{..}} o_{..} \frac{\sum_{i=1}^{m} \sum_{j=1}^{m} o_{ij} \delta_{ij}}{\sum_{i=1}^{m} o_i \sum_{j=1}^{m} o_j \delta_{ij}} = \frac{1-\pi}{o_{..}}
\end{aligned}$$

□

Proof of Proposition 15 (p. 100). Let us calculate:

$$\begin{aligned} P_e^{\pi} &= \sum_{k=1}^{m} {r_k}^2 = \sum_{k=1}^{m} \left(\frac{p_k + q_k}{2} \right)^2 = \sum_{k=1}^{m} \frac{{p_k}^2 + 2p_k q_k + {q_k}^2}{4} \\ &= \sum_{k=1}^{m} \frac{{p_k}^2 - 2p_k q_k + {q_k}^2 + 4p_k q_k}{4} = \sum_{k=1}^{m} \frac{(p_k - q_k)^2 + 4p_k q_k}{4} \\ &= \sum_{k=1}^{m} \left(\frac{p_k - q_k}{2} \right)^2 + \sum_{k=1}^{m} p_k q_k \qquad (*) \\ &\geq \sum_{k=1}^{m} p_k q_k = P_e^{\kappa} \end{aligned}$$

Hence,

$$\pi = \frac{P_o - P_e^{\pi}}{1 - P_e^{\pi}} \leq \frac{P_o - P_e^{\kappa}}{1 - P_e^{\pi}} \leq \frac{P_o - P_e^{\kappa}}{1 - P_e^{\kappa}} = \kappa.$$

The inequality following Equation $(*)$ is justified by the fact that a squared real quantity is never negative, and thus their sum is also never negative.

For the proof of the second part, notice that the first term of Equation $(*)$ is zero if and only if $p_k = q_k$. In that case, the equation becomes

$$P_e^{\pi} = \sum_{k=1}^{m} p_k q_k = P_e^{\kappa}$$

from which the second part of the proposition follows. □

Proof of Proposition 17 (p. 100). Let us calculate:

$$\begin{aligned} P_e^{AC_1} &= \frac{1}{m-1} \sum_{k=1}^{m} r_k (1 - r_k) = \frac{1}{m-1} \sum_{k=1}^{m} \left(r_k - {r_k}^2 \right) = \frac{1}{m-1} \left(\sum_{k=1}^{m} \frac{o_k}{o_{\cdot\cdot}} - \sum_{k=1}^{m} {r_k}^2 \right) \\ &= \frac{1}{m-1} \left(\frac{1}{o_{\cdot\cdot}} \sum_{k=1}^{m} o_k - \sum_{k=1}^{m} {r_k}^2 \right) = \frac{1}{m-1} \left(\frac{o_{\cdot\cdot}}{o_{\cdot\cdot}} - \sum_{k=1}^{m} {r_k}^2 \right) \\ &= \frac{1}{m-1} \left(1 - \sum_{k=1}^{m} {r_k}^2 \right) = \frac{1 - P_e^{\pi}}{m-1} = \frac{D_e^{\pi}}{m-1} \end{aligned}$$

□

Proof of Proposition 18 (p. 100). Starting from Propositions 10 and 12, let us calculate:

$$\begin{aligned} P_e^{\pi} - P_e^{AC_1} &= \left(\frac{1}{m} + \sum_{k=1}^{m} \tilde{r}_k^2 \right) - \left(\frac{1}{m} - \frac{1}{m-1} \sum_{k=1}^{m} \tilde{r}_k^2 \right) \\ &= \frac{1}{m} + \sum_{k=1}^{m} \tilde{r}_k^2 - \frac{1}{m} + \frac{1}{m-1} \sum_{k=1}^{m} \tilde{r}_k^2 = \sum_{k=1}^{m} \tilde{r}_k^2 + \frac{1}{m-1} \sum_{k=1}^{m} \tilde{r}_k^2 \\ &= \left(1 + \frac{1}{m-1} \right) \sum_{k=1}^{m} \tilde{r}_k^2 = \frac{m}{m-1} \sum_{k=1}^{m} \tilde{r}_k^2. \end{aligned}$$

The proposition then follows. □

Proof of Proposition 19 (p. 100). It is sufficient to show that $P_e^\kappa = 1$ (or, equivalently, that $D_e^\kappa = 0$) if and only if there is some $i \in \{1, \ldots, m\}$ such that $p_i = q_i = 1$.

Assume first that there is some $i \in \{1, \ldots, m\}$ such that $p_i = q_i = 1$. Since $\sum p_i = \sum q_i = 1$, it must be that $p_j = q_j = 0$ for all $j \in \{1, \ldots, m\}$ such that $i \neq j$. Thus, $P_e^\kappa = \sum_{k=1}^m p_k q_k = p_i q_i = 1$.

Assume then that $D_e^\kappa = 0$. Thus, $\sum_{i=1}^m \sum_{j=1}^m p_i q_j \delta_{ij} = 0$. Since all p_i, q_j, and δ_{ij} are nonnegative, $p_i q_j \delta_{ij} = 0$ must hold for all $i, j \in \{1, \ldots, m\}$. Since $\delta_{ij} = 1$ for all $i \neq j$, it must hold that $p_i = 0$ or $q_j = 0$ whenever $i \neq j$. Therefore, there is at most one $i \in \{0, \ldots, m\}$ for which $p_i \neq 0$ or $q_i \neq 0$. But then, there must be exactly one i such that $p_i = q_i = 1$. □

Proof of Proposition 20 (p. 100). Recall that

$$D_e^\alpha = \frac{o_{..}}{o_{..} - 1} D_e^\pi \qquad \text{and} \qquad D_e^\pi = \sum_{i=1}^m \sum_{j=1}^m r_i r_j \delta_{ij}.$$

That both are undefined or neither are undefined follows directly from the fact that D_e^α is D_e^π multiplied by an always defined nonzero factor.

Now, the two statistics are undefined if and only if $D_e^\pi = 0$, that is, if and only if $\sum_{i=1}^m \sum_{j=1}^m r_i r_j \delta_{ij} = 0$. Now, an argument very similar to the previous proof shows that this is equivalent to the statement that there is some $i \in \{1, \ldots, m\}$ such that $r_i = 1$. □

Proof of Proposition 21 (p. 100). Notice:

$$\begin{aligned} D_e^{AC_1} = 0 &\iff P_e^{AC_1} = 1 \iff \frac{D_e^\pi}{m-1} = 1 \iff \frac{1}{m-1}\left(1 - \sum_{k=1}^m {r_k}^2\right) = 1 \\ &\iff 1 - \sum_{k=1}^m {r_k}^2 = m - 1 \iff \sum_{k=1}^m {r_k}^2 = 1 - (m-1) \\ &\iff \sum_{k=1}^m {r_k}^2 = 2 - m \end{aligned}$$

Since $\sum_{k=1}^m {r_k}^2$ is always positive and $2 - m$ is never positive, $D_e^{AC_1} \neq 0$. □

Proof of Proposition 22 (p. 100). In all three cases, it is sufficient to show that $P_o - P_e = 0$ if and only if the indicated identity holds. In the first two cases, this follows nearly trivially from the respective definitions of P_o and P_e. In the third case, let us calculate:

$$\begin{aligned} P_o - P_e^\alpha &= \sum_{i=1}^m r_{ii} - \frac{1}{o_{..} - 1}\left(o_{..} \sum_{i=1}^m {r_i}^2 - 1\right) = \sum_{i=1}^m r_{ii} - \sum_{i=1}^m \frac{o_{..}}{o_{..} - 1} {r_i}^2 - \frac{1}{o_{..} - 1} \\ &= \sum_{i=1}^m \left(r_{ii} - \frac{o_{..}}{o_{..} - 1} {r_i}^2\right) - \frac{1}{o_{..} - 1}. \end{aligned}$$

□

Proof of Proposition 23 (p. 101). Let us calculate:

$$\begin{aligned} P_o - P_e^{AC_1} &= P_o - \frac{1}{m-1} D_e^{\pi} = \sum_{i=1}^{m} r_{ii} - \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} r_i r_j \delta_{ij} \\ &= \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} r_{ii} \delta_{ij} - \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} r_i r_j \delta_{ij} \\ &= \frac{1}{m-1} \sum_{i=1}^{m} \sum_{j=1}^{m} \left(r_{ii} - r_i r_j \right) \delta_{ij} \end{aligned}$$

The proposition follows straightforwardly. □

Proof of Proposition 24 (p. 101). Let $m = 2$. Then

$$0 = \sum_{i=1}^{m} \sum_{j=1}^{m} \left(r_{ii} - r_i r_j \right) \delta_{ij} = r_{11} - r_1 r_2 + r_{22} - r_1 r_2$$

and thus $r_{11} + r_{22} = 2 r_1 r_2$, from which the proposition follows. □