Concept Analysis in Programming Language Research

Done Well It Is All Right

Antti-Juhani Kaijanaho
University of Jyvaskyla
Faculty of Information Technology
University of Jyvaskyla, Finland
antti-juhani.kaijanaho@jyu.fi

Abstract

Programming language research is becoming method conscious. Rigorous mathematical or empirical evaluation is often demanded, which is a good thing. However, I argue in this essay that concept analysis is a legitimate research approach in programming languages, with important limitations. It can be used to sharpen vague concepts, and to expose distinctions that have previously been overlooked, but it does not demonstrate the superiority of one language design over another. Arguments and counter-arguments are essential to successful concept analysis, and such thoughtful conversations should be published more.

CCS Concepts • **Software and its engineering** → *General programming languages*; • **General and reference** → General literature;

Keywords programming language research, non-empirical research, research methodology, concept analysis, philosophy, argumentation

ACM Reference Format:
Antti-Juhani Kaijanaho. 2017. Concept Analysis in Programming Language Research: Done Well It Is All Right. In *Proceedings of 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!'17).* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3133850.3133868

1 Introduction

Traditionally, programming language research has not been very self-conscious about research methodology. This is slowly changing, in that the premier venues like OOPSLA are requiring rigorous validation, and some authors (including me) are pushing for the wider acceptance of human-factors research [e. g., 35, 71, 91, 97]. It seems to me that this methodological awakening is a bit too focused on the traditional Humean duality—

> "When we run over libraries, persuaded of these principles, what havoc must we make? If we take in our hand any volume [...] 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."
> — David Hume [39], last paragraph, emphasis in the original

—that is, only mathematical and empirical reasoning[1] are permitted in the halls of science. Except for such dogmatism, I too join in the push for human-factors empirical research.

My goal in this essay is to highlight another worthy research approach, one that has been used in this field since before there were actual computers and is still commonly used. I speak of concept analysis, or the philosophical analysis of concepts.[2] Very rarely do people call attention to the fact that they are taking that approach, and sometimes this lack of explicit discussion of the methodology confuses the authors or the readers into thinking that they are doing something else. When authors are confused, they make claims that are not warranted by their argument. When readers are confused, they think the paper reports bad research when it merely needs to be presented better.

Onward!'17, October 25–27, 2017, Vancouver, Canada

ACM ISBN 978-1-4503-5530-8/17/10...$15.00
https://doi.org/10.1145/3133850.3133868

[1] When Hume was writing, the modern concept of a controlled experiment and the modern disputes among empirical researchers had not been invented yet, so Hume's "experimental reasoning" should not be read to exclude qualitative research.

[2] Note that I am *not* talking about the phases of software development that are sometimes called "analysis" or "conceptual design", discussed by, e. g., Jackson [42]. Nor am I talking about the lattice-theoretical "formal concept analysis" originally proposed by Wille [102]. It is unfortunate that the same words have similar but crucially different meanings in the same field. It is all the more confusing that the meanings are not totally unrelated.

Here is what I mean by concept analysis:

- *Concept analysis means taking a vague concept and proposing a sharper replacement.* Sometimes the result of the analysis is instead that the original concept is incoherent and must be abandoned. The classic example of the former is the analysis of mechanical calculation by Turing [99]. A well-known example of the latter is the analysis of inheritance in statically typed languages by Cook et al. [15], where they showed that the then-predominant idea of conflating inheritance with subtyping is incoherent.
- *Concept analysis also means taking other people's analyses and subjecting them to exacting scrutiny.* When an analysis does not hold up under scrutiny, it would benefit the field if the scrutineer would present a thoughtful counter-argument, and perhaps even their own analysis of the same concept. This ought to repeated until a rough fixed point (though not necessarily an agreement) is reached. Sadly, this sort of productive scholarly discussion appears to be all but absent in the literature of our field.

When distinguishing these meanings is needed, I will call them *analysis as doctrine* and *analysis as dialogue*.

In this essay, I will first argue that conceptual questions have been asked and answered in this field for decades, and that mathematical and empirical methods do not suffice. I will introduce the basic philosophical ideas regarding concepts in Section 2. I will show that questions regarding concepts have been asked in all seriousness from the beginning of our field to the present day; the argument in Subsection 3.1 accomplishes this by pointing out examples from the literature from the 1930s to the present day. I will then argue, in Sections 3.2 and 3.3, that those questions cannot be answered using mathematical or empirical methods.

I will then establish the suitability of philosophical analysis as both doctrine and dialogue for answering conceptual questions. In Section 4, I will argue that philosophical concept analysis backed by an argument can answer conceptual questions, but that it requires a practice of critical dialogue, not just the publication of separate analyses. I will then, in Section 5, endorse previously published proposals (with my amendments) for the assessment of philosophical essays in the programming language field. Finally, in Section 6, I will discuss what contribution concept analysis can and cannot make.

This essay is best viewed as belonging in *methodology*—the interdisciplinary field that studies the way research ought to be conducted, both in general and as applied to particular studies (see, e. g., Cecez-Kecmanovic [11], Hammersley [34]). The field of application I am discussing is programming language research, though my ideas may also be relevant to other subfields of computer science and software engineering. Because of the subject matter, I draw heavily in this essay on both the philosophy and the social science literature.

Methodological works in programming language research are, to the best of my knowledge, fairly rare. Recently several authors have published methodological critiques of this field (see, e. g., Hanenberg [35], Markstrum [64], Stefik and Hanenberg [91, 92], Stefik et al. [93]) but of actual positive methodological essays I am aware of only two. Hanenberg [36] recently published an introduction to controlled experimentation in programming language research. In my dissertation [46], I articulated a research approach of philosophical concept analysis which I then used to explicate the concept of evidence-based programming language design; but while my discussion of the approach was detailed, it was reportedly rather difficult to follow and also lacked a detailed argument in its favor. This essay follows up on the basic idea of the dissertation, that of philosophical analysis as a research approach, and argues for its adoption (as one approach among many) in programming language research. My principal contribution beyond that dissertation is the placement of the approach in the wider context of research methodology, and the argument I present in its support.

In the field of software engineering, methodological discussions have a longer pedigree. There are detailed methodological expositions of at least case studies [88], controlled experiments [104], systematic reviews [54], action research [79], and grounded theory [38, 94]. The information systems and human-computer interaction research fields have their own methodological traditions (see, e. g., Järvinen [44], Lazar et al. [56]). Most relevant to this essay, however, is a recent philosophical concept analysis in software engineering by Dittrich [19]; it is not a methodological work but it argues, similarly to this essay, that philosophical argument can be an appropriate research approach. I base my discussion of quality criteria on Dittrich's; but this essay goes much further than Dittrich's paper in articulating and defending the research approach in detail.

2 Concepts

The literature on concepts is enormous, written over several millennia, and I cannot do justice to it here; my discussion is by necessity rather simplistic. For an overview of the relevant philosophical literature, see the article on concepts in the Stanford Encyclopedia of Philosophy [62] or in the Internet Encyclopedia of Philosophy [20]. The latter is written for the nonspecialist audience and elides much of the complexity.

2.1 The Concept of Concepts

As a first approximation, we can regard concepts as classification devices. For example, we have the concept of *book* that classifies all material objects as being books or not being books. Similarly, in the programming languages field, we

have the concept of *object-oriented programming language*, which classifies programming languages as object-oriented or not. This extends even to singleton concepts: *Phosphorus* classifies material things as being the morning star or not (and it turns out only Venus qualifies).

Concepts are important because we think and communicate using them, and we ascribe values to them. For example, as I am writing this on July 5, 2017, the Rust language web site at https://www.rust-lang.org/ declares that

> "Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety."

This describes Rust in terms of technical concepts like "systems programming language", "segfault-preventing", "thread safety"; the apparent intent of the writer is that Rust is classified positively by these concepts, and I venture to say that the writer intends the reader to consider this a good thing. Whether these are true statements about Rust depends not only on Rust but also on what the concepts "systems programming language" and "thread safety" actually are. The conceptual question here is far from trivial.

It would, however, be wrong to hold that concepts are just sets or classes, in the set-theoretic sense. The classic example, due to Frege [27], concerns the singleton set that contains Venus but corresponds to more than one concept: *Phosphorus* is Venus as seen in the morning sky, and *Hesperus* is Venus as seen in the evening sky, but it would be wrong to say that these are the same concept (which they would be if the set was all that mattered). It is customary to call the set or class associated with a concept its *extension*; whatever it is that distinguishes two co-extensional concepts is then called a concept's *intension* (see, e. g., Chalmers [12] and Rapaport [85]). Similarly, it is wrong to say that the unicorn and the tooth fairy are the same concept even though they have the same (empty) extension; again, it is the intension that differs.

Since concepts are used for communication and thinking, there is a third aspect to them: *designators*. Each concept must be named by some linguistic expression, sometimes more than one, and its name is significant in that it can suggest the intension of the concept.

2.2 Universals versus Social Constructions

The nature of concepts is itself an unsettled matter. It seems natural to me to suppose that there is a real (although intangible) object that the numeral 2 denotes. It is, of course, possible to deny this and hold, for example, that there are no numbers (as distinct from symbols used in calculation and other linguistic elements) and we only make the idea up to explain in our heads how certain formal systems behave. Similarly, one can hold that there is a real (but intangible) object that the word "type" denotes when we are talking about programming language theory, or one can hold that "type" is merely a word that we use to explain the behavior of certain formal systems. Call the first position *realist* and the "real" concepts *universals*; the second position can be called *formalist*. For a formalist, conceptual questions are nonsensical—they merely "arise from our failure to understand the logic of our language" (Wittgenstein [103], Proposition 4.003)

Further, one can deny that, for example, euros on a bank account exist and hold that they are a fiction we make up to explain (or maintain) our society. This is quite plausible in my view, as money in these days is *fiat money*—that is, it is not backed by anything independently valuable like gold, like it used to be. Yet, it is very hard to deny that money in the bank is generally treated as real and as good as (or nowadays, with governments frowning on untraceable transactions, better than) cash. In practical terms, then, money in the bank *is real*; if one needs to pacify an inner objection, one can add that this reality is a mere metaphor or a model. Because this reality is qualitatively different from our ordinary physical reality, we might talk of *social reality*.

Now, a social reality (including money in the bank) is quite literally created by people interacting. When I buy groceries and pay using my debit card, the cashier acting for the store owner accepts it (and consequently my money in the bank) as equal or more in value than the groceries I buy. But my money in the bank is valuable only because the cashier, and everybody else, treats it as valuable. If we collectively decided to ignore bank money, it would become worthless. This is what is meant when people talk of the (social) reality being *socially constructed* (see generally, e. g., Berger and Luckmann [4], Hacking [32], Searle [90]).

It is perfectly possible for a concept to have some features grounded in the material reality and acquire a social construction on top. A well known example comes from sociology: there are undeniable biological differences between human beings that are generally used to classify people as *man* or *woman*, but there are many features of these concepts that are not necessary consequences of those biological differences; thus, while the concepts of man and woman undoubtedly have some grounding in material reality, most of what they are is socially constructed (see, e. g., Berkowitz et al. [5], Lorber [61], West and Zimmerman [101]).

In the social sciences, a claim of social construction is usually multifaceted (with the fourth and fifth facets being optional) [32]: *first,* the target concept is generally seen as natural and unchangeable; *second,* the target concept in fact is not natural but constructed by humans; *third,* the target concept could have been constructed differently, or it could have never existed; *fourth,* the target concept is morally wrong in its current shape; and *fifth,* the target concept should be modified or abolished. For example, the concepts of man and woman have been attacked in just this manner [69].

The programming language context brings a twist to these philosophical and social theoretical concerns. Traditional philosophy aims to describe the objective reality beyond that which physics and the other natural sciences are able to

tell us (hence, *meta*physics, which is a branch of philosophy focusing on the nature of the reality); similarly, the social sciences aim to describe (and often transform) the social reality. However, programming language research largely is not interested in what is out there—it is about what is not there yet, but could be. While there are some constraints imposed by reality on the design of programming languages—the theory of computation and other finite mathematics being the most important, and the psychology of programmers also having some influence—most of programming language design is what it says on the tin: *design*. Programming languages are constructed by humans, and most of the abstract concepts related to them are quite obvious social constructions.

3 Conceptual Questions

In this section, I will introduce a class of research questions and argue that these questions cannot be answered using conventional research techniques—they are neither mathematical nor empirical in nature.

3.1 Examples

I will start by recounting three important concepts in computing science that have been (or currently are) subject to controversy. The first takes us to a time before there were programming languages or even computers. In the early 20th Century, mathematicians were struggling with profound questions. One important issue was to figure out what it means for something to be a formal system; as Gödel [30] noted in a 1964 postscriptum to his 1934 lecture notes, this required clarifying the concept of a mechanical procedure (or the equivalent concept of effective calculability or computability). After all, the "essence" of a formal system "is that reasoning is completely replaced by mechanical operations" [30, p. 370]. Multiple answers were offered in the mid-1930s:

- Church [13] proposed both the lambda calculus (due jointly to himself and his student Kleene) and the notion of general recursion (which he attributed jointly to Herbrand and Gödel) as equivalent definitions of effective calculability;
- Turing [99] proposed, independently of Church, his computing machines, now called Turing machines, as a definition of computability.
- Post [84] specified, independently of Turing, a model of computation very similar to Turing's, and conjectured it to be formally equivalent to general recursion.

Both Church and Turing explicitly claimed to have found a precise concept to replace the intuitive but vague concepts used in the previous literature. They thus answered the conceptual question, what does it mean to calculate mechanically? Both answers are now generally accepted as adequate.

My second example is a bundle of concepts that is still extremely important, namely *object-orientation*. The phrase *object-oriented programming* was coined by Alan Kay in around 1967 as the name of a fairly specific concept of programming—live cells communicating by messages, with no explicit data present—inspired by and largely based on existing ideas, including those of the Simula 67 programming language [50, 51]. Eventually *object-oriented* became a concept classifying not only ways of programming but various programming languages and even processor architectures[3], but it became increasingly opaque what it actually means. Like an early writer predicted [86, p. 51], "Everyone will be in favor of it. Every manufacturer will promote his products as supporting it. Every manager will pay lip service to it. Every programmer will practice it (differently). And no one will know just what it is." Some years later, another writer discussed the nature of OO in an essay called "My Cat Is Object-Oriented" [53].

About 25 years later, another writer lamented that "we do not yet thoroughly understand the fundamental concepts that define the OO approach" [2, p. 123]. Soon after, nearly a decade ago, Cook [14] argued that the textbook definition of objects is conceptually wrong. By then, as now, object-oriented programming was a topic taught to university freshmen in all information technology disciplines, and its core has thus essentially ossified; yet, the same conceptual issues continue to haunt those very students [105] and even inspires research [74, 75]. The conceptual questions like "What are objects?" and "What is inheritance?" remain relevant.

My third example is still a major research question. Exactly when did *type* become a concept (as opposed to just a word used in its ordinary meaning) in the field of programming languages, remains unclear despite several authors recently having investigated the history [46, 52, 65, 66]. One possible account starts with the (ramified) theory of types proposed a century ago by Russell [89], which eventually begat the simple theory of types which evolved into the typed lambda calculus all programming language theorists know and love. Another account starts from the word "type" used in the Fortran manuals and the specification of Algol 60. Regardless of which historical account one accepts, it is clear that the concept of *type* is alive and controversial: apart from the context of a single specific language, where one can get authoritative answers from a language standard or a reference implementation, there is no consensus answer to such questions as "What is a type?" or even whether "dynamically typed" is a meaningful or a nonsensical term.

3.2 They Are Not Mathematical

It seems obvious to consider two of my three examples as mathematical questions. The *answer* to "what is a mechanical procedure" or "what is a type", at least, seems mathematical:

[3] Although the iAPX 432 was described as "object-based", it was made clear that this was meant to include the "object-oriented" property; see [41].

Turing machines, the lambda calculus, the theory of general recursion, the simple theory of types, and System F (to mention some examples) all are indisputably mathematical objects of great ingenuity and importance, and we certainly expect research papers introducing type systems to discuss them in a very mathematical way (stating their essence as formal systems and proving the usual soundness theorems). Yet I claim that the *questions* and the *method for answering* them are not mathematical at all.

To the extent that mathematics has a method, it must be the *axiomatic method*. It is a method of exposition: the completed mathematical theory is presented as starting from certain fundamental assumptions and developing through a series of definitions and proved lemmas, and culminating in a central theorem, and perhaps, starting from new definitions, repeating ad nauseam. It is not the way mathematics is actually made (for discussion, see, e. g., [22, 55, 96]). For the actual process of coming up with this axiomatically presented theory, there is no method that I am aware of. Yet, for present purposes, a method of exposition is quite enough.

Suppose that I were to try to answer the question "what is a type" using the axiomatic method. To start, I would have to set up axioms. Classically, as Hutton and Gregory [40] phrased it, "*An Axiom, or Maxim,* is a self-evident proposition; requiring no formal demonstration to prove its truth; but received and assented to as soon as mentioned." (p. 2, emphasis in the original). Disregarding that this view of axioms is obsolete, where would I find self-evident propositions regarding the "type" concept that would be assented to by everyone? Surely if there are such things, they would have been found already, and the matter would no longer be alive

The modern mathematician tends to view axioms simply—in the words of Feferman [23]—as "definitions of kinds of structures which have been recognized to recur in various mathematical situations" (p. 403). But when trying to settle the concept of "type" I am not dealing with the abstraction of a recurring structure from particular instances into a general theory; or at least, it is precisely the question of what abstraction is best chosen that is at issue. Thus, I find no help here either.

However, there is one additional viewpoint to mathematical axioms that needs to be considered. Feferman [23] called a certain axioms "fundamental": these are "axioms for such fundamental concepts as number, set and function that underlie *all* mathematical concepts" (p. 403, emphasis in the original). Easwaran [21] argues, convincingly to me, that such axioms are a way for mathematicians to insulate mathematics from philosophy: mathematicians can decide individually on what philosophical basis they adopt them, but once they have each done that, they can work together to explore the mathematical reality they establish. Indeed, Easwaran argues that presuppositions move from explicitly stated assumptions to unstated axioms once they become generally accepted. Now, even this way of looking at axioms denies me: in order to set up axioms, I need to convince everyone that they are the right ones.

The mathematical content of any theorems proved is necessarily implicit already in the axiom system; the statement and proof of a theorem explores the *necessary consequences* of the choice of axioms, giving the researcher, at most, the opportunity to discover that they made wrong turns somewhere earlier, and must revise (as Lakatos [55] discusses). Whether the axioms truly describe the concept that the researcher wishes them to describe is beyond the reach of mathematics, and like Easwaran argues, is a matter for philosophy to decide.

Thus it appears that these are not mathematical questions, nor can they be answered using mathematical methods. In the words of Turing [99, p. 249]:

> "…to show that the "computable" numbers include all numbers which would naturally be regarded as computable. All arguments which can be given are bound to be, fundamentally, appeals to intuition, and for this reason rather unsatisfactory mathematically."

3.3 They Are Not Empirical

At the most abstract level, we may regard research as a process generating *answers* to *questions*, or evaluating the *truth value* of *propositions*. These two characterizations are clearly linked, as a question together with an answer is a proposition, and any proposition can be viewed as a question together with the proposed answer "yes".

Philosophers have distinguished between *a priori* and *a posteriori* propositions at least since the 18th Century and Kant [49], based on whether sense observations are required to evaluate it: *a priori* propositions can be evaluated using reason alone, from the armchair if you will, while *a posteriori* propositions require some observation of the real world to evaluate. For example, it is *a priori* true that $1 + 1 = 2$, but it is *a posteriori* false that the Earth is a flat disk.

In scientific[4] practice, it is more common to use the adjective *empirical*, which refers to obtaining knowledge by observing the reality using the senses, and derives from the ancient Greek adjective ἐμπειρία (empeiria), 'experienced' [76], describing an ancient school of physicians who relied on personal and collective experience instead of theoretical reasoning [26, 82]. We can speak of empirical questions and empirical propositions, meaning *a posteriori* questions and propositions; similarly, we speak of *empirical research*, meaning the use of sense observation to generate scientific results (or, as one anonymous reviewer put it, making the results "be

[4] In this essay, I use "science" and its variants in a broad sense, including physics, computing, humanities, and social science, roughly coextensively with "scholarship" and "research". The issue of demarcation—distinguishing science from pseudoscience (cf. Popper [83] and Pigliucci and Boundry [81])—is beyond the scope of this essay.

based on data"). Conversely, there are questions and propositions that are *not* empirical, meaning that they are *a priori*.

There are two possible fundamental routes to my claim that these conceptual questions are not empirical, depending on one's ontological commitments, and one practical route.

One could hold that there are universals such as redness or the number 42 that are independent of time and place; or that the programming concepts of *computability*, *objects*, and *types* are universals, existing independently of time, space, and us humans. Such universals—being independent of spacetime—cannot be perceived by the senses, and thus they cannot be empirical.

One could also take a constructionist view. The idea here is that concepts are created by humans, as they use them in discussions. Here the question of empiricality is trickier. Certainly once a constructionist concept has been generally accepted, that is, once there is a *social construction* of that concept [4, 32], it becomes an empirical question to determine what that social construction is. However, that empirical question is mostly of interest to educators and outside researchers (such as those in the field of science and technology studies).

For us who are participants in this field—the insiders—the more important question, from a constructionist point of view, is, how *should* we construct these concepts. At that point, we are no longer asking questions about things that exist in any reality, but making decisions about what to create in the future. No sense experience can, in general, answer questions about things that do not (yet) exist. Further, sense experience can reveal only what *is*; to move from that *is* to *should* requires a nonempirical principle; thus, while empirical considerations influence the answer to these questions, they cannot alone decide them.

One does not, of course, have to commit to universals or constructionism *in toto*; it is perfectly rational to regard some concepts as universals and some other concepts as (social) constructions. For example, personally I am inclined to view computability as a universal concept while objects and types are, in my view, best regarded as constructions (though not necessarily, at this point, social constructions).

Another way to think about this issue is reflect on the various kinds of empirical methodology. The gold standard for empirical evidence, the randomized controlled experiment, is not suited for answering conceptual questions; the best it can do is to measure the indirect effects of adopting various conceptual models. However, it is certainly possible, at least in principle if not in practice, to ascertain the social construction (if any) of a concept by the means of surveys (either of the literature, as was done by, e. g., Armstrong [2] and Jordan et al. [45], or of the relevant social groups, though I am not aware of anyone having done that in our field), but this does not answer the interesting question of whether this construction is in some sense the correct or the best one.

4 Methodology

In this section, I will explicate and defend the research approach of philosophical concept analysis for answering conceptual questions. I must, however, first give some background.

Methodology requires, among other things, defining the goals of (particular kinds of) research, and arguing that certain ways of conducting research fulfill those goals, perhaps with caveats. As such, methodology is closely connected to philosophy, particularly *ontology* (the theory of the nature of the reality) and *epistemology* (the theory of knowledge).

We can categorize research by *research approaches* (see, e.g., Vessey et al. [100]), roughly corresponding to what some writers (e. g., Lincoln and Guba [59]) call *research paradigms*. Research approaches differ from each other in their *ontological* (what is the nature of reality), *epistemological* (what is the nature of knowledge), *methodological* (how does one go about generating knowledge), and *axiological* (what knowledge is valuable) assumptions [59, p. 37]. More concretely, research may be guided by a *research method*,[5] which is "a specific technique or design used to conduct a study" [100, fn. 1 on p. 248]; each research approach tends to favor particular methods, though the relationship is not bijective.

Ontologically, one can make a distinction between different *realities*. This word choice probably seems too grand and even preposterous, but it is standard usage in this context (see, e. g., Moon and Blackman [70]). There are three categories of reality I wish to point out: the physical reality, the social realities, and what I would tentatively call the software reality. The *physical reality* is a familiar concept: as I, a physics layman, currently understand it, it contains matter and energy and has four spacetime dimensions. A *social reality* consists of institutions that some specific collection of people interacting together agree to exist (see, e. g., Berger and Luckmann [4], Searle [90]); since groups of people can disagree, there may be multiple social realities. Finally, the *software reality* consists of all the programs and data stored in computer storage media, including all the currently running instances of programs. When research methodologists talk of *ontology*, they mean a theory of reality in this sense.

At the highest level of abstraction, we can distinguish between empirical and non-empirical research approaches, based on whether they deal in *a posteriori* or *a priori* knowledge, respectively. One common empirical approach that Järvinen [44, p. 10] calls *theory-testing*, Vessey et al. [100, p. 251] call *evaluative–deductive*, and many writers (e. g., Guba and Lincoln [31], Lincoln et al. [60], Nekrašas [73]) call *positivist*, transports the research approach dominant in empirical physics to the study of social reality: it assumes

[5] Research methods should be distinguished from the concept of *the scientific method*, which "contains firm, unchanging, and absolutely binding principles for conducting the business of science" [24, p. 7]. There are good reasons to think that there is no such thing (cf. Feyerabend [24] and Kaijanaho [47]).

that there is an single, objective social reality, independent of individuals, which can be reliably captured by using the human senses as augmented by measurement devices. This approach favors the controlled experiment and aims to generate universally applicable laws that can be used for prediction and control.

Another approach, called *constructivist* by Lincoln and Guba [59] (previously *naturalistic*, see [58]) and *evaluative–interpretive* by Vessey et al. [100, p. 251], explicitly rejects the model of physics for studying human society and posits that there are multiple social realities, defined by particular (groups of) people, and that each reality can only be captured by interacting by the defining people (which quite possibly changes that reality). This approach favors ethnography and other forms of qualitative inquiry, and aims to generate faithful descriptions of the realities under examination.

A third approach, called *critical theory* by Guba and Lincoln [31] and *evaluative–critical* by Vessey et al. [100, p. 251], assumes that there is a common social reality which was constructed in the past and has, over time, ossified and become apparently objective and real for most intents and purposes; the goal of critical theory is to expose them as the changeable constructs that they are, and to take action to transform such reality into something the researcher regards more ethical. Critical theory favors qualitative methods and aims to generate changes in the social reality.

The computing disciplines have developed an additional empirical research approach not derived from the social science traditions. Here, the reality of interest is the software reality, and knowledge is generated by the means of examining or running programs. The most prominent method here is *computational experiments*—the study of algorithms by exposing their implementations to a wide variety of automatically generated stimuli and measuring the effort expended by the implementation as a function of stimulus parameters [6, 28, 29, 37, 48, 67, 72]; it is relevant to programming language research mostly in the study of implementation techniques.

There are also multiple non-empirical research approaches. Discussing the information systems field, Hamilton and Ives [33] distinguished conceptual research from other nonempirical research (e. g., tutorials and reviews—though I would categorize well-done literature reviews as empirical), and Alavi and Carlson [1] listed conceptual, illustrative, and applied concepts as sub-classes of non-empirical research. Vessey et al. [100, p. 251], in their unified taxonomy of computing research, list two classes of non-empirical research approaches, that of *descriptive* (including system descriptions and literature reviews) and *formulative* (including framework, guideline, model, taxonomy, and concept formulation) research approaches. Some other writers, for example Järvinen [43, 44] and Hanenberg [35], only credit one non-empirical approach, that of mathematical (including stochastic theoretical) research.

Historically, there was a very influential non-empirical research approach that is generally labeled as *rationalism*: it was claimed either that it is possible to learn truths about reality by intuition and deduction or that we humans possess innate knowledge about the reality that we can uncover by reasoning [63]. Let me be clear that I do not advocate this sort of rationalism in this essay.

4.1 Philosophical Concept Analysis

All discussion of methodology must start from the basic assumptions of what sort of reality and what aspects of it (*ontology*) are of interest, and what sort is the knowledge about them that is of interest (*epistemology*). Only from explicit consideration of these fundaments can we derive any kind of principles of methodology for a particular discipline and research approach.

For concept analysis in programming language research, the objects of interest are concepts that classify things relevant to programming. Of interest are the software reality (regarding technological artefacts such as programming languages) and the social reality (regarding programmers and their interaction); it is quite possible that some concepts span both kinds of reality.

The epistemological issue was already broached earlier: conceptual questions cannot be answered by either mathematical or empirical methods. It is a trickier issue what *can* be used, and it is not irrational to conclude that they cannot be answered at all. It is my intention in this section, however, to argue that they can be answered, though not with any sort of certainty of correctness, using philosophical concept analysis.

I will now state a high-level definition:[6]

Definition: *A philosophical concept analysis is a claim, supported by argument, that one concept should be replaced by another concept.*

There are two main variants:

- A *classical analysis* (see, e. g., McGinn [68]), holds that these concepts are equivalent, but one of them (the *analysandum*) is a vague preexisting concept and the other (the *analysans*) is, it is claimed, more precise and often novel.
- A *Carnapian explication*, suggested by Carnap [10], holds that one of the concepts (the *explicandum*) should be replaced by the other (the *explicatum*) because the latter is a precise and in also other ways better alternative; but no equivalence is claimed.

In both variants, the analysans or explicatum will usually be specified intensionally, by giving necessary and sufficient conditions, and an analysans or explicatum is intended to be

[6] For an overview of the extensive and multiple millennia spanning literature on this, see the article on analysis in the Stanford Encyclopedia of Philosophy [3].

usable as a stipulated definition in future work (to allow, so to speak, the mathematics—or empirical work—to begin).

What Turing did to computability is clearly an example of conventional concept analysis: he took a vague concept (computability) and provided a precisely defined equivalent (computability by Turing machine), together with a compelling argument supporting their equivalence (this is discussed in more detail by, e. g., Davis [17, p. 14] and Kaijanaho [46, p. 54–55]). Similarly, we can regard the various formal type systems published in the literature as proposed (Carnapian) explications of the concept of type (but often without an accompanying argument supporting it); recently, Kell [52] offered a clear analysis of the concept with an argument. Conversely, Petricek [80] argues powerfully that there is no (nor should there be a) single analysis (or, using his terms, "definition") of the concept.

In the context of object-orientation, a radical (Carnapian) re-explication that rejected object classes and their inheritance, replacing them with prototype objects and delegation, was discussed by Borning [8] and Lieberman [57]. Both papers are clearly concept analyses and offer strong arguments in support of the central claims.

There are two issues to dispose: *First:* Is a concept analysis, understood this way, an answer to a conceptual question? *Second:* How (or to what extent) can we demonstrate that a concept analysis is correct?

The first issue is easy: a question of the form "what is X" is certainly answered by the classical analysis "X is Y"—whether it is an interesting answer is a separate issue that does not belong under methodology. The case of a Carnapian explication is trickier, but if it is established that the explicatum truly is a better alternative to the explicandum, the explication does answer the question.

I now turn to the issue of establishing the correctness of an analysis or explication. As I have already argued, an appeal to mathematics or empirical data will not work. At the same time, an *ipse dixit* is equally unpersuasive, at least to anyone looking at the matter critically. What is needed is something in between. The traditional tool in the philosophical practice is *argumentation*.

4.2 Argumentation

In informal logic,[7] an *argument* consists of a proposition (the *conclusion* of the argument) together with one or more other propositions offered as *reasons* to accept the conclusion, where those reasons *support* the conclusion (see, e. g., Blair [7, p. 189] or Fisher [25]). A good argument, according to Blair [7], is one whose reasons are individually acceptable to its audience and together (taking into account the structure of the argument) sufficient to support the conclusion.

These criteria are largely not assessable by using the tools of formal logic—only in some cases will the argument have a form that is deductively valid, and even then the question of acceptability of the reasons remains. More often it is possible to identify missing reasons that would transform a deductively invalid argument into a valid one, but an argument critique that takes this step risks critiquing a strawman instead of the argument intended.

A particular common move in modern analytical philosophy is sometimes called a *thought experiment*, *intuition pump*, or the *method of cases*. Here, the philosopher sets up a concrete but hypothetical (and sometimes obviously counterfactual) scenario and tells its story with an intended obvious moral. For example, Turing [99, p. 249–250] reasons that his machine can do everything that a human computer can by inviting the reader to imagine a human computer at work, and then transforming that image in ways that—as the reader can easily agree—do not affect the capability of the computer, so long as human fallibility is discounted, and eventually reaching the machine model we now call Turing machines. Similarly, Strachey [95] and Reynolds [87] argue for the need for parametric polymorphism by discussing the case of the *map* (in the case of Strachey) or the *sort* (Reynolds) function; Reynolds then continues to argue for a specific design of the polymorphic lambda calculus, which can be regarded as an explication of Strachey's vague concept of a polymorphic type system.

It is sometimes appropriate to use empirical or mathematical results as reasons in a philosophical argument. It is, however, important to remember that since the questions are not empirical, the argument must have more than empirical data backing it. For example, there is a major difference between the empirical claim that the (in my case imaginary) interviewees view objects as data records with associated procedures and the philosophical claim that objects are data records with associated procedures. There is no inference rule justifying the move from an empirical "is" to a philosophical "ought".

4.3 Standard of Correctness

Consider the standard for when a conceptual analysis is correct. In the case of a universal concept which is independent of spacetime and people (assuming such concepts even exist), all we can hope to have is justified beliefs. An argument can provide justification for a belief. This justification becomes stronger if there are multiple arguments, and particularly if counterarguments are successfully rebutted. The maximum possible justification is achieved if there is a rational agreement of all relevant people. Similarly, in the case of social constructions, a concept analysis is correct if it is accepted as a (new) social construction by the relevant social group; this requires the agreement of all the relevant people. In

[7] Informal logic has its roots in the ancient times, but its modern development started in the 1950s following the publication of the seminal works by Perelman and Olbrechts-Tyteca [78] and Toulmin [98], and furthered, among other things, by developments in teaching argumentation to university students in the 1960s (see, e. g., Blair [7], p. 185–186). It is the philosophy arm of the interdisciplinary field of argumentation theory.

both cases, the standard of correctness is thus the same as the standard for objectivity in science in general (see, e. g., Popper [83]): intersubjective agreement.

This sort of intersubjective agreement is not guaranteed by argument, since one can always dispute the reasons given (their *modus ponens* is your *modus tollens*, as a philosophers' famous saying goes). It can also be achieved by irrational means, e. g., through indoctrination, but such success cannot be credited to the analysis. However, argument can (when used well) create intersubjective agreement: Turing's argument regarding computation is a very good example.

The intersubjective agreement angle suggests another very important aspect to the methodology of concept analysis: it needs a practice of critical dialogue. It is not possible to delay the publication of an analysis until intersubjective agreement is demonstrated, for testing for such agreement requires the prior publication of the analysis. It is only after publication that we can learn whether the analysis is correct or not, by the criterion of intersubjective agreement. If (and when) there are problems identified in the analysis, these need to be pointed out, so that the original position can be either refined or abandoned.[8] The literature of concept analysis thus becomes a conversation.

5 Assessment of a Concept Analysis Essay

As concept analysis is not mathematical or empirical in nature, we should not demand rigorous mathematical proofs or careful controlled experiments from essays presenting these analyses. Similarly, the criteria developed for assessing empirical work—whether quantitative (internal and external validity, see Campbell [9]) or qualitative (credibility, transferability, dependability, and confirmability; see Lincoln and Guba [58])—are concerned with the relationship of the empirical data used to the conclusions, and thus are completely inapplicable to concept analysis to the extent that it does not employ original empirical research in developing reasons in the argument.

But neither is engaging in concept analysis a license to publish anything whatsoever. It is not so that "anything goes" [24]; even Feyerabend himself did not deny the value of discipline-level standards. There are standards in concept analysis, vague and admittedly subjective though they are.

Dittrich [19, p. 221] proposed to evaluate philosophical works in software engineering by "rigour of argumentation" and "relevance of results". In my dissertation [46, p. 57], endorsing these broad criteria, I further proposed to evaluate rigor (following Paseau [77]) by whether reasons are stated explicitly and by the extent to which the steps made in arguments are small; but "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" [46, p. 57].

I still agree with these proposals. Relevance is always important, in all fields and all methods. But once relevance is achieved, there is still much room for both brilliance and drivel. Since it is not reasonable to expect a concept analysis to be irrefutable, the optimal level of clarity and rigor ought to be that which best allows the discussion to continue thoughtfully. As excessive rigor is often counterproductive toward that goal, this requires, as Paseau [77] argues, that an argument is made as rigorous as necessary but no more. I find it impossible to give general rules delineating that point, save from the obvious: be rigorous enough to be understood, and not so rigorous that you are not understood.

I will add one further criterion. Reports of concept analysis should be good scholarship; that is, the argument should at minimum acknowledge and at best engage seriously and thoughtfully with previous analyses as well as relevant non-concept analysis research. Where disagreement exists, the analysis report should develop a thoughtful counterargument. The goal of a discussion is frustrated if nobody listens to others.

In addition to the criteria for argumentation, the evaluation of the analysis or explication being defended deserves consideration as well. A useful starting point seems to me to be the Carnap [10, p. 7] criteria for explication: the explicatum should be a suitable replacement for the explicandum, and additionally exact, fruitful (in terms of provoking further research), and simple. That the explicatum fulfills these criteria should, of course, be defended by the argument offered, but any reviewer should also make their own independent assessment regardless of the merits of the argument.

These are all external criteria that are hard for the author to self-analyze before submission. However, one useful exercise for the author (beyond the obvious technique of soliciting private feedback from peers) is to take the role of the devil's advocate and try to attack their own argument with the best counter-arguments they can come up with. The essay will be stronger once those counter-arguments are properly dealt with in the text itself.

One potential criterion I would completely reject. One might think that not being convinced by the argument would be sufficient grounds for rejecting an analysis. It is not. The question is rather, does the analysis and its supporting argument advance the discussion even if it is wrong. This philosophical attitude is well displayed in the following anonymous referee comment reported by the philosopher John Danaher [16]:

> "This is a good paper. In the opinion of this reviewer, it is wrong at nearly every important point, but it is wrong in ways that are interesting and important – a genuine contribution to the philosophical discussion."

[8] This critical dialogue is analogous to the publication of replication attempts of empirical research. For the same reasons, such critical discussion needs to be encouraged in both empirical and nonempirical research.

Of course, the essay is a literary form, and writing a good essay requires more than just presenting a rigorous and relevant argument with good scholarship. However, such artistic considerations are beyond my competence to analyze, and I will say no more of them.

Finally, I do not mean to suggest that concept analysis must be accepted in all venues or that it must be funded by grants. My position is merely that it cannot be rejected simply because it is concept analysis, or because someone might see concept analysis as lacking in rigor as a general matter. Specific analyses can be vulnerable to methodological criticism (including the lack of rigor), and all publication venues and all grant agencies have standards that go beyond methodology; for example, the surprise factor that makes a claim interesting (cf. Davis [18]) is a common criterion beyond methodological correctness.

6 Contributions and Non-contributions

I will be blunt here. There are many things that concept analysis does *not* contribute to. Hanenberg [35, 36] is quite right that answering questions regarding usefulness to real humans in the real world requires empirical work. Attempting to use philosophical arguments to advance human-factors claims is foolish. Similarly, a philosophical concept analysis provides no guarantees of internal consistency, and thus a philosophical argument cannot be effective in support of any type system soundness claim.

Similarly, it is a foolish thing for a philosophical argument to assert itself as the final answer on its topic, or for any reader to cite it as a source of definitive authority. There is always room for disagreement in philosophy and concept analysis.

What concept analysis does contribute is greater clarity in concepts. Sometimes it will expose fatal flaws in concepts previously thought to be sound, and sometimes it will demonstrate that a particular concept is actually ambiguous and needs to be split into multiple concepts.

Concept analysis matters even to empirical research. When one is trying to conceive a controlled experiment to measure the relative ranking of, say, object-oriented programming language paradigm and functional programming language paradigm, or static and dynamic typing, one must decide how to operationalize these concepts. A rather naïve approach, but dominant in the literature, is to choose a representative language from each paradigm or typing discipline (or to design representative languages for the purposes of the experiment).

But what justifies generalizing from those languages to the paradigms? One could simply decline to argue the point, beyond possibly noting it as a limitation of the study (and this is a perfectly rational response for a Popperian), but I find this quite unsatisfying. The question becomes: what could possibly be offered as a serious argument in support? I can imagine two contenders: *First,* one could assert definitions for the paradigms or typing disciplines that make the problem go away; for example, defining OO as Smalltalk and FP as Haskell. But that merely means that if I do not agree with those definitions, the study becomes utterly irrelevant for me; it is essentially the same move that mathematics makes when it postulates axioms. *Second,* one can offer (or adopt previously published) analyses of the terms.

This second option is why concept analysis is not just relevant but necessary for controlled experiments. Concepts and their analysis are directly relevant to and potentially dispositive of construct validity and thus of external validity.

Above all, concept analysis is *necessary*. We cannot avoid defining the concept of type in type systems work, but if we simply state a definition by fiat, we are essentially working hypothetically: if you, dear reader, accept my definition, then you will benefit from my work; otherwise, never mind. To move from hypotheticals into assertions of fact, we need to support our definitions with an analytical argument; we may be wrong, but at least we will not be hypothetical.

7 Conclusion

There is a place for concept analysis in the toolbox of programming language researchers. Done correctly and for the right reasons, it can contribute significantly to our field.

Denying concept analysis its place in the toolbox has two possible outcomes. On the one hand, perhaps researchers will heed that prohibition and avoid concept analysis in the future. But then, our concepts will be developed by accident, memetic mutation, and authoritarian decrees. On the other hand, perhaps researchers will use concept analysis despite its shunning; but in those circumstances, it must be done stealthily, disguised as other kinds of research. Such dishonesty would not bode well for our research community.

I hold that concept analysis belongs here. Perhaps you disagree. If you do, I hope to read your counterargument in a published essay soon.

Acknowledgments

I first presented versions of these claims in my doctoral dissertation [46], though the details and my arguments have evolved since then. Accordingly, thanks are due to Tommi Kärkkainen, Vesa Lappalainen, and Ville Tirronen (my doctoral advisors); Matthias Felleisen and Andreas Stefik (the external reviewers), and Lutz Prechelt (my opponent in the dissertation defense). My thinking on these issues has been influenced by discussions, in particular, with Stefan Hanenberg, Ville Isomöttönen, and Maija Tuomaala, as well as the participants of the Dagstuhl Seminar 15222. It should be noted that these people do not in all cases share my views on these issues. The anonymous reviewers gave very useful feedback, which has helped me improve this essay quite a

bit. Portions of the thinking reported here was done while I was visiting the University of Duisburg–Essen in early 2015.

References

[1] Maryam Alavi and Patricia Carlson. 1992. A Review of MIS Research and Disciplinary Development. *Journal of Management Information Systems* 8, 4 (1992), 45–62.

[2] Deborah J. Armstrong. 2006. The Quarks of Object-Oriented Development. *Commun. ACM* 49, 2 (2006), 123–128. https://doi.org/10.1145/1113034.1113040

[3] Michael Beaney. 2016. Analysis. In *The Stanford Encyclopedia of Philosophy* (summer 2016 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University, Stanford, CA. https://plato.stanford.edu/archives/sum2016/entries/analysis/

[4] Peter J. Berger and Thomas Luckmann. 2011. *The Social Construction of Reality: A Treatise in the Sociology of Knowledge.* Open Road, New York.

[5] Dana Berkowitz, Namita N. Manohar, and Justine E. Tinkler. 2010. Walk Like a Man, Talk Like a Woman: Teaching the Social Construction of Gender. *Teaching Sociology* 38, 2 (2010), 132–143. https://doi.org/10.1177/0092055X10364015

[6] Stephen M. Blackburn, Kathryn S. McKinley, Robin Garner, Chris Hoffmann, Asjad M. Khan, Rotem Bentzur, Amer Diwan, Daniel Feinberg, Daniel Frampton, Samuel Z. Guyer, Martin Hirzel, Antony Hosking, Maria Jump, Han Lee, J. Eliot B. Moss, Aashish Phansalkar, Darko Stefanovik, Thomas VanDrunen, Daniel von Dincklage, and Ben Wiedermann. 2008. Wake Up and Smell the Coffee: Evaluation Methodology for the 21st Century. *Commun. ACM* 51, 8 (Aug. 2008), 83–89. https://doi.org/10.1145/1378704.1378723

[7] J. Anthony Blair. 2012. *Groundwork in the Theory of Argumentation.* Number 21 in Argumentation Library. Springer, Dordrecht. https://doi.org/10.1007/978-94-007-2363-4

[8] A. H. Borning. 1986. Classes Versus Prototypes in Object-Oriented Languages. In *ACM '86 Proceedings of 1986 ACM Fall joint computer conference.* IEEE Computer Society, Los Alamitos, CA, 36–40. http://dl.acm.org/citation.cfm?id=324493.324538

[9] Donald T. Campbell. 1957. Factors Relevant to the Validity of Experiments in Social Settings. *Psychological Bulletin* 54, 4 (1957), 297–312. https://doi.org/10.1037/h0040950

[10] Rudolf Carnap. 1962. *Logical Foundations of Probability* (2 ed.). University of Chicago Press, Chicago.

[11] Dubravka Cecez-Kecmanovic. 2011. On Methods, Methodologies and How They Matter. ECIS 2011 Proceedings. (2011). http://aisel.aisnet.org/ecis2011/233/

[12] David J. Chalmers. 2002. On Sense and Intension. *Noûs—Philosophical Perspectives* 36, s16 (2002), 135–182. https://doi.org/10.1111/1468-0068.36.s16.6

[13] Alonzo Church. 1936. An Unsolvable Problem of Elementary Number Theory. *American Journal of Mathematics* 58, 2 (1936), 345–363. https://doi.org/10.2307/2371045

[14] William R. Cook. 2009. On Understanding Data Abstraction, Revisited. In *OOPSLA '09 Proceedings of the 24th ACM SIGPLAN conference on Object oriented programming systems languages and applications.* ACM, New York, 557–572. https://doi.org/10.1145/1640089.1640133

[15] William R. Cook, Walter L. Hill, and Peter S. Canning. 1990. Inheritance Is Not Subtyping. In *POPL '90 Proceedings of the 17th ACM SIGPLAN–SIGACT symposium on Principles of programming languages.* ACM, New York, 125–135. https://doi.org/10.1145/96709.96721

[16] John Danaher. 2015. How I Write for Peer Review. (Feb. 2015). Retrieved April 22, 2017 from http://philosophicaldisquisitions.blogspot.fi/2015/02/how-i-write-for-peer-review.html

[17] Martin Davis. 1982. Why Gödel Didn't Have Church's Thesis. *Information and Control* 54, 1–2 (1982), 3–24. https://doi.org/10.1016/S0019-9958(82)91226-8

[18] Murray S. Davis. 1971. That's Interesting! Towards a Phenomenology of Sociology and a Sociology of Phenomenology. *Philosophy of the Social Sciences* 1, 2 (1971), 309–344. https://doi.org/10.1177/004839317100100211

[19] Yvonne Dittrich. 2016. What does it mean to use a method? Towards a practice theory for software engineering. *Information and Software Technology* 70 (2016), 220–231. https://doi.org/10.1016/j.infsof.2015.07.001

[20] Dennis Earl. no date. Concepts. Internet Encyclopedia of Philosophy. (no date). Retrieved 2017-07-05 from http://www.iep.utm.edu/concepts/

[21] Kenny Easwaran. 2008. The Role of Axioms in Mathematics. *Erkenntnis* 68, 3 (2008), 381–391. https://doi.org/10.1007/s10670-008-9106-1

[22] Johannes Emerich. 2016. How Are Programs Found: Speculating about Language Ergonomics with Curry–Howard. In *Onward! 2016 Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software.* ACM, New York, 212–223. https://doi.org/10.1145/2986012.2986030

[23] Solomon Feferman. 2000. Why the Programs for New Axioms Must Be Questioned. *Bulletin of Symbolic Logic* 6, 4 (2000), 401–413. https://doi.org/10.2307/420965

[24] Paul Feyerabend. 2010. *Against Method* (4 ed.). Verso, London.

[25] Alec Fisher. 1988. *The Logic of Real Arguments.* Cambridge University Press, Cambridge.

[26] Michel Frede. 1987. *Essays in Ancient Philosophy.* University of Minnesota Press, Minneapolis.

[27] Gottlob Frege. 1948. Sense and Reference. *Philosophical Review* 57, 3 (1948), 209–230. https://doi.org/10.2307/2181485 Translated from the German "Über Sinn und Bedeutung" (1892) by Max Black.

[28] Ian P Gent, Stuart A. Grant, Ewen MacIntyre, Patrick Prosser, Paul Shaw, Barbara M Smith, and Toby Walsh. 1997. *How Not To Do It.* Research Report 97.27. University of Leeds, School of Computer Studies. Retrieved 2017-06-28 from https://www.imbe.leeds.ac.uk/computing/research/publications/reports/1997/1997_27.pdf

[29] Andy Georges, Dries Buytaert, and Lieven Eeckhout. 2007. Statistically Rigorous Java Performance Evaluation. In *OOPSLA '07 Proceedings of the 22nd annual ACM SIGPLAN conference on Object-oriented programming systems and applications.* ACM, New York, 57–76. https://doi.org/10.1145/1297105.1297033

[30] Kurt Gödel. 1986. On undecidable propositions of formal mathematical systems (1934). In *Kurt Gödel – Collected Works – Volume 1 (Publications 1929–1936)*, Solomon Feferman, John W. Dawson, Jr., Stephen C. Kleene, Gregory H. Moore, Robert M. Soloway, and Jean van Heijenoort (Eds.). Oxford University Press, New York, 346–371.

[31] Egon G. Guba and Yvonna S. Lincoln. 1994. Competing Paradigms in Qualitative Research. In *Handbook of Qualitative Research*, Norman K. Denzin and Yvonna S. Lincoln (Eds.). SAGE, Thousand Oaks.

[32] Ian Hacking. 1999. *The Social Construction of What?* Harvard University Press, Cambridge, MA.

[33] Scott Hamilton and Blake Ives. 1982. MIS Research Strategies. *Information & Management* 5, 6 (1982), 339–347. https://doi.org/10.1016/0378-7206(82)90033-7

[34] Martyn Hammersley. 2011. *Methodology: Who Needs It?* SAGE, London. https://doi.org/10.4135/9781446287941

[35] Stefan Hanenberg. 2010. Faith, Hope, and Love: An essay on software science's neglect of human factors. In *OOPSLA '10 Proceedings of the ACM international conference on Object oriented programming systems languages and applications.* ACM, New York, 933–946. https://doi.org/10.1145/1932682.1869536

[36] Stefan Hanenberg. 2017. Empirical, Human-Centered Evaluation of Programming and Programming Language Constructs: Controlled Experiments. In *Tutorial Lectures of the Grand Timely Topics in Software Engineering: International Summer School GTTSE 2015 (Lecture*

Notes in Computer Science), Jácome Cunha, João P. Fernandes, Ralf Lämmel, João Saraiva, and Vadim Zaytsev (Eds.). Springer, Cham, 45–72. https://doi.org/10.1007/978-3-319-60074-1_3

[37] Scott Hazelhurst. 2010. Truth in advertising: reporting performance of computer programs, algorithms and the impact of architecture and systems environment. *South African Computer Journal* 46 (2010), 24–37. https://doi.org/10.18489/sacj.v46i0.50

[38] Rashina Hoda, James Noble, and Stuart Marshall. 2011. Grounded Theory for Geeks. In *Proceedings of the 18th Conference on Pattern Languages of Programs (PLoP), PLoP'11,*. ACM, New York, Article 24, 17 pages. https://doi.org/10.1145/2578903.2579162

[39] David Hume. 2011. *An Enquiry Concerning Human Understanding*. Project Gutenberg, Salt Lake City. http://www.gutenberg.org/ebooks/9662

[40] Charles Hutton and Olinthus Gregory. 1836. *A Course of Mathematics* (11 ed.). Vol. 1. Gilbert & Livington, London. https://play.google.com/store/books/details?id=0a4TAAAAQAAJ

[41] Intel 1981. *Introduction to the iAPX 432 Architecture*. Intel. http://bitsavers.trailing-edge.com/pdf/intel/iAPX_432/171821-001_Introduction_to_the_iAPX_432_Architecture_Aug81.pdf

[42] Daniel Jackson. 2015. Towards a Theory of Conceptual Design for Software. In *Onward! 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. ACM, New York, 282–296. https://doi.org/10.1145/2814228.2814248

[43] Pertti Järvinen. 2008. Mapping Research Questions to Research Methods. In *Advances in Information Systems Research, Education and Practice (IFIP International Federation for Information Processing)*, David Avison, George M. Kasper, Barbara Pernici, Isabel Ramos, and Dewald Roode (Eds.). Springer, Boston, 29–41. https://doi.org/10.1007/978-0-387-09682-7-9_3

[44] Pertti Järvinen. 2012. *On Research Methods*. Opinpajan kirja, Tampere.

[45] Howell Jordan, Goetz Botterweck, John Noll, Andrew Butterfield, and Rem Collier. 2015. A feature model of actor, agent, functional, object, and procedural programming languages. *Science of Computer Programming* 98 (2015), 120–139. https://doi.org/10.1016/j.scico.2014.02.009

[46] Antti-Juhani Kaijanaho. 2015. *Evidence-Based Programming Language Design—A Philosophical and Methodological Exploration*. Number 222 in Jyväskylä studies in computing. University of Jyväskylä, Jyväskylä. http://urn.fi/URN:ISBN:978-951-39-6388-0 PhD diss.

[47] Antti-Juhani Kaijanaho. 2015. Ramblings inspired by Feyerabend's Against Method, Part II: My preliminary take. (Oct. 2015). Retrieved 2017-06-28 from http://antti-juhani.kaijanaho.fi/newblog/archives/1979

[48] Tomas Kalibera and Richard Jones. 2013. Rigorous Benchmarking in Reasonable Time. In *ISMM '13 Proceedings of the 2013 international symposium on memory management*. ACM, New York, 63–74. https://doi.org/10.1145/2464157.2464160

[49] Immanuel Kant. 1996. *Critique of Pure Reason*. Hackett, Indianapolis. Translated from the German "Kritik der reinen Vernunft" (published in 1781 and 1787) by Werner S. Pluhar.

[50] Alan Kay and Stefan Ram. 2003. Dr. Alan Kay on the Meaning of "Object-Oriented Programming". (2003). Retrieved April 13, 2017 from http://www.purl.org/stefan_ram/pub/doc_kay_oop_en

[51] Alan C. Kay. 1996. The Early History of Smalltalk. In *History of Programming Languages–II*, Thomas J. Bergin, Jr. and Richard G. Gibson, Jr. (Eds.). ACM Press, New York, 511–598. https://doi.org/10.1145/234286.1057828

[52] Stephen Kell. 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*. ACM, New York, 227–241. https://doi.org/10.1145/2661136.2661154

[53] Roger King. 1989. My Cat Is Object-Oriented. In *Object-Oriented Concepts, Databases, and Applications*, Won Kim and Frederick H. Lochovsky (Eds.). ACM, New York, 23–30.

[54] Barbara Ann Kitchenham, David Budgen, and Pearl Brereton. 2016. *Evidence-Based Software Engineering and Systematic Reviews*. CRC, Boca Raton.

[55] Imre Lakatos. 1976. *Proofs and Refutations: The Logic of Mathematical Discovery*. Cambridge University Press, Cambridge.

[56] Jonathan Lazar, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. *Research Methods in Human–Computer Interaction* (2 ed.). Morgan Kaufmann, Cambridge, MA.

[57] Henry Lieberman. 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*. ACM, New York, 214–223. https://doi.org/10.1145/28697.28718

[58] Yvonna S. Lincoln and Egon G. Guba. 1985. *Naturalistic Inquiry*. SAGE, Newbury Park, CA.

[59] Yvonna S. Lincoln and Egon G. Guba. 2013. *The Constructivist Credo*. Left Coast, Walnut Creek, CA.

[60] Yvonna S. Lincoln, Susan A. Lynham, and Egon G. Guba. 2011. Paradigmatic Controversies, Contradictions, and Emerging Confluences, Revisited. In *The SAGE Handbook of Qualitative Research* (4 ed.), Norman K. Denzin and Yvonna S. Lincoln (Eds.). SAGE, Los Angeles.

[61] Judith Lorber. 1994. *Paradoxes of Gender*. Yale University Press, New Haven, Chapter "Night to His Day": The Social Construction of Gender, 13–36.

[62] Eric Margolis and Stephen Laurence. 2014. Concepts. In *The Stanford Encyclopedia of Philosophy* (spring 2014 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University, Stanford, CA. https://plato.stanford.edu/archives/spr2014/entries/concepts/

[63] Peter Markie. 2015. Rationalism vs. Empiricism. In *The Stanford Encyclopedia of Philosophy* (summer 2015 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University, Stanford, CA. https://plato.stanford.edu/archives/sum2015/entries/rationalism-empiricism/

[64] Shane Markstrum. 2010. Staking Claims: A History of Programming Language Design Claims and Evidence: A Positional Work in Progress. In *Proceeding PLATEAU '10 Evaluation and Usability of Programming Languages and Tools*. ACM, New York, Article 7, 5 pages. https://doi.org/10.1145/1937117.1937124

[65] Simone Martini. 2016. Several Types of Types in Programming Languages. In *History and Philosophy of Computing: Third International Conference, HaPoC 2015, Pisa, Italy, October 8–11, 2015, Revised Selected Papers (IFIP Advances in Information and Communication Technology (IFIPAICT))*, Fabio Gadducci and Mirko Tavosanis (Eds.). Springer, Cham, 216–227. https://doi.org/10.1007/978-3-319-47286-7_15

[66] Simone Martini. 2016. Types in Programming Languages, Between Modelling, Abstraction, and Correctness. In *Pursuit of the Universal: 12th Conference on Computability in Europe, CiE 2016, Paris, France, June 27 – July 1, 2016, Proceedings (Lecture Notes in Computer Science)*, Arnold Beckmann, Laurent Bienvenu, and Nataša Jonoska (Eds.). Springer, Cham, 164–169. https://doi.org/10.1007/978-3-319-40189-8_17

[67] Catherine C. McGeoch. 1996. Toward an Experimental Method for Algorithm Simulation. *INFORMS Journal on Computing* 8, 1 (1996), 1–15.

[68] Colin McGinn. 2012. *Truth by Analysis: Games, Names, and Philosophy*. Oxford University Press, New York.

[69] Mari Mikkola. 2017. Feminist Perspectives on Sex and Gender. In *The Stanford Encyclopedia of Philosophy* (summer 2017 ed.), Edward N. Zalta (Ed.). Metaphysics Research Lab, Stanford University, Stanford, CA. https://plato.stanford.edu/archives/sum2017/entries/feminism-gender/

[70] Katie Moon and Deborah Blackman. 2014. A Guide to Understanding Social Science Research for Natural Scientists. *Conservation Biology* 28, 5 (2014), 1167–1177. https://doi.org/10.1111/cobi.12326

[71] Brad A. Myers, Andreas Stefik, Stefan Hanenberg, Antti-Juhani Kaijanaho, Margaret Burnett, Franklyn Turbak, and Philip Wadler. 2016. Usability of Programming Languages Special Interest Group (SIG) Meeting at CHI 2016. In *CHI EA '16 Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*. ACM, New York, 1104–1107. https://doi.org/10.1145/2851581.2886434

[72] Todd Mytkowicz, Amer Diwan, Matthias Hauswirth, and Peter F. Sweeney. 2009. Producing Wrong Data Without Doing Anything Obviously Wrong!. In *ASPLOS XIV Proceedings of the 14th international conference on Architectural support for programming languages and operating systems*. ACM, New York, 265–276. https://doi.org/10.1145/1508244.1508275

[73] Evaldas Nekrašas. 2016. *The Positive Mind: Its Development and Impact on Modernity and Postmodernity*. Central European University Press, Budapest.

[74] James Noble. 2009. The Myths of Object-Orientation. In *ECOOP 2009—Object-Oriented Programming—23rd European Conference—Genoa, Italy, July 6–10, 2009—Proceedings (Lecture Notes in Computer Science)*, Sophia Drossopoulou (Ed.). Springer, Berlin, 619–629. https://doi.org/10.1007/978-3-642-03013-0_29

[75] James Noble, Andrew P. Black, Kim B. Bruce, Michael Homer, and Mark S. Miller. 2016. The Left Hand of Equals. In *Onward! 2016 Proceedings of the 2016 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software*. ACM, New York, 224–237. https://doi.org/10.1145/2986012.2986031

[76] Oxford English Dictionary 2014. empiric, *n.* and *adj.* (March 2014). Retrieved June 21, 2017 from http://www.oed.com/view/Entry/61340

[77] A. C. Paseau. 2016. What's the Point of Complete Rigour? *Mind* 125, 497 (2016), 177–207. https://doi.org/10.1093/mind/fzv140

[78] Chaïm Perelman and L. Olbrechts-Tyteca. 1969. *The New Rhetoric: A Treatise on Argumentation*. University of Notre Dame Press, Notre Dame. Translated from the French "La Nouvelle Rhétorique: Traité de l'Argumentation" (1958) by John Wilkinson and Purcell Weaver.

[79] Kai Petersen, Cigdem Gencel, Negin Asghari, Dejan Baca, and Stefanie Betz. 2014. Action Research as a Model for Industry–Academia Collaboration in the Software Engineering Context. In *WISE '14 Proceedings of the 2014 international workshop on Long-term industrial collaboration on software engineering*. ACM, New York, 55–62. https://doi.org/10.1145/2647648.2647656

[80] Tomas Petricek. 2015. Against a Universal Definition of 'Type'. In *Onward! 2015 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software (Onward!)*. ACM, New York, 254–266. https://doi.org/10.1145/2814228.2814249

[81] Massimo Pigliucci and Maarten Boundry. 2013. *Philosophy of Pseudoscience: Reconsidering the Demarcation Problem*. University of Chicago Press, Chicago.

[82] Gianna Pomata. 2011. A Word of the Empirics: The Ancient Concept of Observation and its Recovery in Early Modern Medicine. *Annals of Science* 68, 1 (2011), 517–538. https://doi.org/10.1080/00033790.2010.495039

[83] Karl R. Popper. 1980. *The Logic of Scientific Discovery*. Unwin Hyman, Boston. Originally published in German as 'Logic der Forschung' in 1934.

[84] Emil L. Post. 1936. Finite Combinatory Processes—Formulation 1. *Journal of Symbolic Logic* 1, 3 (1936), 103–105. https://doi.org/10.2307/2269031

[85] William J. Rapaport. 2012. Intensionality vs. Intentionality. (March 2012). Retrieved 2017-07-06 from https://www.cse.buffalo.edu//~rapaport/intensional.html

[86] Tim Rentsch. 1982. Object Oriented Programming. *ACM SIGPLAN Notices* 17, 9 (1982), 51–57. https://doi.org/10.1145/947955.947961

[87] John C. Reynolds. 1974. Towards a Theory of Type Structure. In *Programming Symposium Proceedings, Colloque sur la Programmation, Paris, April 9–11, 1974 (Lecture Notes in Computer Science)*, B. Robinet (Ed.). Springer, Berlin, 408–425. https://doi.org/10.1007/3-540-06859-7_148

[88] Per Runeson, Marting Höst, Austen Rainer, and Björn Regnell. 2012. *Case Study Research in Software Engineering: Guidelines and Examples*. Wiley, Hoboken, NJ.

[89] Bertrand Russell. 1908. Mathematical Logic as based on the Theory of Types. *American Journal of Mathematics* 30, 3 (1908), 222–262. https://doi.org/10.2307/2369948

[90] John R. Searle. 2006. Social Ontology: Some Basic Principles. *Anthropological Theory* 6, 1 (2006), 12–29. https://doi.org/10.1177/1463499606061731

[91] Andreas Stefik and Stefan Hanenberg. 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*. ACM, New York, 283–299. https://doi.org/10.1145/2661136.2661156

[92] Andreas Stefik and Stefan Hanenberg. 2017. Methodological Irregularities in Programming-Language Research. *Computer* 50, 8 (2017), 60–63. https://doi.org/10.1109/MC.2017.3001257

[93] Andreas Stefik, Stefan Hanenberg, Mark McKenney, Anneliese Andrews, Srinivas Kalyan Yellanki, and Susanna Siebert. 2014. What is the Foundation of Evidence of Human Factors Decisions in Language Design? An Empirical Study on Programming Language Workshops. In *ICPC 2014 Proceedings of the 22nd International Conference on Program Comprehension*. ACM, New York, 223–231. https://doi.org/10.1145/2597008.2597154

[94] Klaas-Jan Stol, Paul Ralph, and Brian Fitzgerald. 2016. Grounded Theory in Software Engineering Research: A Critical Review and Guidelines. In *ICSE '16 Proceedings of the 38th International Conference on Software Engineering*. ACM, New York, 120–131. https://doi.org/10.1145/2884781.2884833

[95] Christopher Strachey. 2000. Fundamental Concepts in Programming Languages. *Higher-Order and Symbolic Computation* 13, 1–2 (2000), 11–49. https://doi.org/10.1023/A:1010000313106 Written in 1967 and widely circulated as a typescript before posthumous publication.

[96] William P. Thurston. 1994. On Proof and Progress in Mathematics. *Bull. Amer. Math. Soc.* 30, 2 (1994), 161–177. https://doi.org/10.1090/S0273-0979-1994-00502-6

[97] Walter F. Tichy. 1998. Should Computer Scientists Experiment More? *Computer* 31, 5 (1998), 32–40. https://doi.org/10.1109/2.675631

[98] Stephen E. Toulmin. 2003. *The Uses of Argument* (updated ed.). Cambridge University Press, New York. First edition published in 1958.

[99] A. M. Turing. 1937. On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society* s2-42, 1 (1937), 230–265. https://doi.org/10.1112/plms/s2-42.1.230

[100] Iris Vessey, V. Ramesh, and Robert L. Glass. 2005. A unified classification system for research in the computing disciplines. *Information and Software Technology* 47, 4 (2005), 245–255. https://doi.org/10.1016/j.infsof.2004.08.006

[101] Candace West and Don H. Zimmerman. 1987. Doing Gender. *Gender & Society* 1, 2 (1987), 125–151. https://doi.org/10.1177/0891243287001002002

[102] Rudolf Wille. 2009. Restructuring Lattice Theory: An Approach Based on Hierarchies of Concepts. In *Formal Concept Analysis: 7th International Conference, Darmstadt, Germany, May 21–24, 2009, Proceedings (Lecture Notes in Artificial Intelligence)*, Sébastien Ferré and Sebastian Rudolph (Eds.). Springer, Berlin, 314–339. https://doi.org/10.1007/978-3-642-01815-2_23 Originally published in 1982.

[103] Ludwig Wittgenstein. 1974. *Tractatus Logico-Philosophicus*. Routledge, London. Translated to English by D. F. Pears and B. F. McGuinness.

[104] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in Software Engineering*. Springer, Heidelberg. https://doi.org/10.1007/978-3-642-29044-2

[105] Stelios Xinogalos. 2015. Object-Oriented Design and Programming: An Investigation of Novices' Conceptions on Objects and Classes. *ACM Transactions on Computing Education* 15, 3, Article 13 (2015), 21 pages. https://doi.org/10.1145/2700519