Science of Computer Programming 156 (2018) 68–89

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

The two paradigms of software development research

Paul Ralph [a,b,*]

[a] University of Auckland, New Zealand
[b] University of British Columbia, Canada

ARTICLE INFO

Article history:
Received 11 April 2017
Received in revised form 30 November 2017
Accepted 9 January 2018
Available online 11 January 2018

Keywords:
Empiricism
Rationalism
Philosophy of science
Software design
Empirical software engineering

ABSTRACT

The most profound conflict in software engineering is not between positivist and interpretivist research approaches or Agile and Heavyweight software development methods, but between the *Rational* and *Empirical Design Paradigms*. The Rational and Empirical Paradigms are disparate constellations of beliefs about how software is and should be created. The Rational Paradigm remains dominant in software engineering research, standards and curricula despite being contradicted by decades of empirical research. The Rational Paradigm views analysis, design and programming as separate activities despite empirical research showing that they are simultaneous and inextricably interconnected. The Rational Paradigm views developers as executing plans despite empirical research showing that plans are a weak resource for informing situated action. The Rational Paradigm views success in terms of the Project Triangle (scope, time, cost and quality) despite empirical researching showing that the Project Triangle omits critical dimensions of success. The Rational Paradigm assumes that analysts elicit requirements despite empirical research showing that analysts and stakeholders co-construct preferences. The Rational Paradigm views professionals as using software development methods despite empirical research showing that methods are rarely used, very rarely used as intended, and typically weak resources for informing situated action. This article therefore elucidates the Empirical Design Paradigm, an alternative view of software development more consistent with empirical evidence. Embracing the Empirical Paradigm is crucial for retaining scientific legitimacy, solving numerous practical problems and improving software engineering education.

1. Introduction

What can be asserted without evidence, can be dismissed without evidence.
–Hitchens' Razor

The interdisciplinary academic discourse on design is bifurcated into two, possibly incommensurable paradigms [1–3], here called *The Rational Design Paradigm* and *The Empirical Design Paradigm*. Consider the conflicting descriptions of software development shown in Box 1.

* Correspondence to: Department of Computer Science, University of Auckland, Auckland, New Zealand.
E-mail address: paul@paulralph.name.

https://doi.org/10.1016/j.scico.2018.01.002
0167-6423/

Box 1
Illustrative metanarratives of the two paradigms.

The rational design paradigm metanarrative	The empirical design paradigm metanarrative
The problem is known and the goal of the system is clear. Analysts elicit comprehensive, unambiguous requirements which are agreed by the client. Designers search a conceptual solution space for design candidates that satisfy the requirements. They use logic and reason to deduce an appropriate architecture or user interface. Design decisions are concentrated in this phase of the project. Developers select an appropriate software development method and use it to build the system. Although perfect rationality is impossible, developers strive to be as structured, methodical and rational as they can. They plan development as a series of activities or phases with milestones and execute this plan. Unexpected events trigger re-planning. Teams understand and evaluate their progress in terms of the plan. The project is successful if it delivers the agreed scope within the allotted time and budget, at a reasonable quality. Researchers understand this process in terms of lifecycle models and software development methods.	There is no "the problem." There is a situation that different stakeholders (with different goals) perceive as problematic in different ways. Analysts work with stakeholders to collaboratively construct ideas and preferences for a possible system. It is not clear which of these ideas and preferences are requirements because no one knows for sure whether each feature is critical, optional, or counterproductive. Understanding of the problematic situation and possible design candidates coevolve: ideas about solutions trigger problem reframing, which triggers new solution ideas, and so on. Designers rely on creativity and intuition. Design pervades the project, with key properties emerging from users, designers and developers during interviews, analysis, programming, refactoring, etc. Each project presents unique sequences of events, which do not necessarily resemble known methods or process models. Unexpected events are common. Plans and software development methods and consequently weak resources for informing behavior, so people improvise. Project success is complicated and controversial but making the problematic situation better in the eyes of its stakeholders generally outweighs technical performance and satisfying contracts. Researchers understand this process in terms of teleological and dialectical process theories, as well as professional behavior.

Some software engineering (SE) researchers and professionals identify more with the story on the left, while others identify more with the story on the right. This creates enormous confusion and conflict in the SE community. Clever, well-informed people disagree because they are using different terms, or using the same terms to mean different things. This paper therefore attempts to unravel, define, explain and examine the two paradigms, and thereby ameliorate ongoing confusion and conflict.

Here, *paradigm* is used in the Kuhnian sense of the "constellation of beliefs, values, techniques, and so on shared by the members of a given community" [4]. Paradigms are not methods; they are not typically used by practitioners directly.

Brooks [5] identified three "formulations" of the Rational Design Paradigm:

1. the mechanical-engineering view of design as a methodical, orderly process, as described by Pahl and Beitz [6];
2. the artificial-intelligence view of design as a search for satisfactory alternatives given goals and constraints, by a designer exhibiting "procedural rationality", as formulated by Simon [7];
3. the managerial view of design as a sequence of weakly-coupled phases, i.e., the Waterfall Model [8] – but more the straw man Royce critiqued than the more iterative model he proposed.

Similarly, at least three formulations of the Empirical Design Paradigm are evident:

1. the view of designer as a "reflective practitioner" alternating between problem framing, adjusting a design concept and evaluating the adjustment's consequences [9];
2. the view of the designer as a creative agent whose attention oscillates between an ill-defined problem concept and a tentative solution concept (coevolution), gradually developing an understanding of both [10–12];
3. the view of design as a political process characterized by interpersonal conflicts, disagreement over goals, politicking and the supremacy of emotional considerations over efficiency (cf. [13–15]).

The conflict between these two paradigms has received little attention in the SE literature, perhaps because it is obscured by: 1) disputes between Agile and Heavyweight (also known as traditional or plan-driven) software development methods [16]; and 2) disputes between positivist and interpretivist epistemological approaches [17]. The Rational/Empirical conflict is more foundational than either of these. The Agile/Heavyweight conflict concerns *which* methods to use; the Rational/Empirical conflict concerns *whether* methods are used or useful. The Positivist/Interpretivist conflict concerns *how* to gather and analyze empirical evidence; the Rational/Empirical conflict concerns *whether* to gather and analyze empirical evidence.

While many have contributed to clarifying the two paradigms (e.g. [1–3,5,9]), their exact compositions and underlying philosophical assumptions remain ambiguous. Moreover, the interpretation and implications of the two paradigms for SE contexts is not well understood. This raises the following research question.

Research Questions: *What are the Rational and Empirical Design Paradigms, their epistemological assumptions and implications for software development?*

Here, *design* refers to specifying the properties of an object by creating a model, prototype or the object itself [18]. Design "encompasses all the activities involved in conceptualizing, framing, implementing, commissioning, and ultimately

modifying complex systems – not just the activity following requirements specification and before programming, as it might be translated from a stylized software engineering process" [19]. (The idea that design is a phase of development is part of the Rational Paradigm; see Section 3.) Meanwhile, *empirical* refers to all research that collects and analyzes qualitative or quantitative data and should not be conflated with *experimental* or *positivist*.

In addressing the above research question, this article makes three contributions:

1. It explains the differing epistemological positions underlying the two paradigms (Section 2).
2. It clearly defines the Rational and Empirical Paradigms with examples of their components (Section 3).
3. It describes the implications of the two paradigms for SE research and practice (Section 4).

Parts of this paper are necessarily argumentative because we cannot empirically evaluate a philosophy that rejects empiricism. We return to this and other limitations in Section 5, which also summarizes the paper's contributions and suggests future research.

2. Design epistemology – Rationalism vs. Empiricism

This paper labels the two paradigms "Rational" and "Empirical" after their differing epistemologies: Rationalism and Empiricism. To understand the source of the conflict between the two paradigms, we first need a broad sketch of these differing epistemologies.

In epistemology, *knowledge* means "justified, true belief."[1] Rationalists and Empiricists fundamentally disagree on what constitutes legitimate justification. Broadly speaking, *Empiricism* is the view that the **only** source of justification is observation. *Rationalism* is the view that some knowledge can be justified by reason and intuition as well as or instead of observation. The Rational Paradigm rests on the philosophy of Rationalism [5].

A comprehensive account of the Rationalism/Empiricism conflict is beyond the scope of this paper, which focuses instead on the basic principles needed to understand the contemporary conflict between the Rational and Empirical Design Paradigms.

2.1. The six theses

Rationalism and Empiricism are subject-specific [20]; for instance, one can simultaneously adopt Rationalism regarding mathematics and Empiricism regarding psychology. Within a subject area, Rationalism and Empiricism are mutually exclusive. One is an empiricist regarding subject area S if one adopts the following thesis:

1. "*The Empiricism Thesis:* We have no source of knowledge in S or for the concepts we use in S other than sense experience" [21 original italics].

In contrast, one is a Rationalist regarding S if one proposes any additional source of knowledge concerning S besides sense experience. Rationalists typically advance some combination of intuition and our rational nature, represented by the following three theses:

2. "*The Intuition/Deduction Thesis*: Some propositions in ... S, are knowable by us by intuition alone; still others are knowable by being deduced from intuited propositions."
3. "*The Innate Knowledge Thesis*: We have knowledge of some truths in a particular subject area, S, as part of our rational nature."
4. "*The Innate Concept Thesis*: We have some of the concepts we employ in a particular subject area, S, as part of our rational nature" [21 original italics].

Many rationalists also adopt two related theses:

5. *"The Indispensability of Reason Thesis*: The knowledge we gain in subject area, S, by intuition and deduction, as well as the ideas and instances of knowledge in S that are innate to us, could not have been gained by us through sense experience."
6. "*The Superiority of Reason Thesis*: The knowledge we gain in subject area S by intuition and deduction or have innately is superior to any knowledge gained by sense experience" [21 original italics].

For example, suppose a population health researcher claims to know that obesity increases the odds of heart attacks based on a 20-year cohort study. The researcher is invoking the empiricism thesis; she is an empiricist regarding this area of population health.

[1] Gettier cases may demonstrate a problem with this definition, but it is not relevant for the purposes of this article.

In contrast, suppose a mathematician claims to know that the Pythagorean theorem is correct based on a proof using the postulates of Euclidean geometry. This mathematician is invoking the Intuition/Deduction Thesis and is therefore a Rationalist regarding geometry. Similarly, suppose a researcher claims to know that gun-ownership leads to more violence during muggings based on a game theory simulation [22]. Like the mathematician, she is invoking The Intuition/Deduction Thesis, and is a rationalist regarding this area of criminology. An economist who claims to know that increasing the money supply will increase inflation based on an economic model rather than empirical data is likewise a rationalist regarding this area of economics.

If the above mathematician, game theorist or economist claims that their theories could not arise from empirical research, they adopt The Indispensability of Reason Thesis. If a dissenter produces apparently contradictory empirical observations and the mathematician, game theorist or economist disregards the empirical data in favor of their logical proofs, they are adopting The Superiority of Reason Thesis. While contemporary articles rarely make such statements, Parnas [23] adopts both The Indispensability of Reason Thesis and The Superiority of Reason Thesis regarding software engineering. He argues for more mathematical proofs since "what can be learned from empirical studies, while important, is very limited" (p. 2) and discounts both experimental (p. 4) and survey research (p. 5) as intrinsically unreliable.

Contemporary examples of The Innate Knowledge Thesis and The Innate Concept Thesis are less common. Universal Grammar Theory posits that certain grammatical structures and the capability to learn grammar are hardwired into humans from birth, but manifest only in children that are exposed to language [24]. Some see this as consistent with the Innate Knowledge and Innate Concept Theses [25]. However, the posited grammatical structures are not exactly knowledge or concepts in the philosophical sense [26].

2.2. Interpretation

Empiricists believe that knowledge can **only** be justified by sense experience. For an empiricist, sensory experience (i.e. observation) is the final arbiter of truth [27]. If common sense and intuition disagree with observation, common sense and intuition are wrong. This does **not** mean that empiricists refuse to use logic or interpret theories and evidence. Rather, empiricists view the outcome of reasoning, intuition, interpretation and so on as conjecture or belief rather than knowledge.

Moreover, empiricists disagree on what *kind* of observation is best. For example, many social scientists believe that social phenomena require different methods of investigation from physical phenomena (e.g. case studies and action research instead of experiments and simulations).

Rationalists meanwhile typically accept that some knowledge is justified by sense experience, but claim that other knowledge is justified by reason or intuition. Empiricism and rationalism are therefore mutually exclusive. Embracing the Empiricism Thesis entails rejecting the credibility of intuition, innate knowledge, innate concepts and the superiority of rational inference.

Different rationalists propose different sources of knowledge. Some believe that knowledge comes from intuition and deduction; others believe knowledge comes from our rational nature; and some accept both sources. Many, but not all, rationalists also believe that some of this non-empirical knowledge cannot be gained from observation or that non-empirical knowledge is more reliable than empirical knowledge.

Obviously, individuals vary greatly in their epistemologies. One can simultaneously adopt different approaches to Rationalism or Empiricism in different domains (e.g. The Innate Concept Thesis in language and hermeneutics in sociology). Similarly, an individual person or text may advance meaningful or self-contradictory hybrids of rationalist and empiricist views.

The Rationalism/Empiricism debate largely occurred in ancient Greece between Plato (a rationalist) and Aristotle (an empiricist) and again in 17th- to 19th-century Europe between rationalists including Descartes, Kant, Leibniz and Spinoza, and empiricists including Berkeley, Hume, Locke and Mill [21,28]. However, some contemporary philosophers, including Robert Brandom, still adopt and defend rationalist positions. More importantly, rationalist rhetoric is still used to justify or attack theories and research in many fields, including mathematics, economics and computer science. Non-science professionals regularly adopt rationalist viewpoints (see Section 4.3); for instance, dismissing counterintuitive empirical evidence for contradicting common sense implicitly adopts the Superiority of Reason Thesis.

2.3. Common misconceptions

Before continuing, two common misconceptions require clarification. Briefly, Empiricism is not equivalent to logical positivism, and The Empiricism Thesis is not a defense of anecdotal evidence.

Empiricism is sometimes conflated with logical positivism (e.g. [29,30]). Logical positivism assumes that theories are supported by unambiguous facts discovered by unbiased observers. Anti-positivism and postmodernism more generally reject the existence of unambiguous facts, and assume that biased observers construct theories from observations, which they interpret based on existing theory. The Empiricism thesis simply says that there are no sources of knowledge beyond observation. Empiricism does not make any claims about the existence of unambiguous facts, the bias of observers or how facts relate to theories. Empiricism predates 20th century philosophical debates between positivism, antipositivism, modernism, postmodernism, constructivism, etc. In the context of the Rationalism/Empiricism conflict, logical positivism, falsificationism, pragmatism, interpretivism, constructivism and Bayesian epistemology all may be viewed as versions of

empiricism [31] just as The Intuition/Deduction, Innate Concept and Innate Knowledge Thesis all may be viewed as versions of Rationalism.

The second common misconception is related to anecdotal evidence. Suppose that a programmer develops several e-commerce websites. Further suppose that the programmer claims to know that these projects use Ruby on Rails based on observing (and writing) the code. This invokes The Empiricism Thesis. The programmer's observations justify this knowledge claim. The programmer does not need a formal study to see which language is used for a project.

However, suppose the programmer claims to know that the websites are successful *because* they use Ruby on Rails. This belief is not justified by the programmer's observations. The programmer has *intuited*, not *observed*, the causal relationship. The programmer is implicitly arguing that it is intuitively obvious that the projects succeeded because of, not in spite of, the language used. This invokes Rationalism – specifically The Intuition/Deduction Thesis. Empiricism holds that knowledge is justified by observation **alone**, not by combinations of observation and intuition.

This becomes especially confusing because people tend simply to justify claims based on personal experience, which sounds like empiricism. People tend not to emphasize the intuitive leaps required to get from their observations to the claims, which reveals their invocation of Rationalism. To summarize: justifying knowledge claims that are not directly observable based on anecdotal evidence invokes Rationalism, not Empiricism.

Having clarified the debate between Rationalism and Empiricism, we now turn to their manifestations in software engineering – the Rational and Empirical Design Paradigms.

3. Deconstructing the rational design paradigm

Precisely defining the Rational and Empirical Paradigms is challenging. However, we can proceed by construction; that is, by discussing a series of interrelated conflicts and then assembling definitions from these conflicts. Beyond their underlying epistemologies discussed in the previous section, at least five major conflicts are evident: 1) model of design; 2) theory of action; 3) views of success and requirements; 4) process models and theories; and 5) practical aim and focus.

3.1. Model of design

In both the Rational and Empirical Paradigms, *design* broadly means determining the properties of an artifact. However, the two paradigms have different "models of design;" that is, views of who determines the properties of the artifact and how they go about it. The Rational Paradigm's model of design is sometimes called *Technical Rationality* while the Empirical Paradigm's model of design is called *Reflection-in-Action*.

Technical Rationality was independently developed in computer science by Herbert Simon, Allen Newell and their collaborators (cf. [7,32]) and in engineering by Gerhard Pahl, Wolfgang Beitz and their collaborators (cf. [6]). Design professionals are modeled as rational agents attempting to optimize a design candidate for known constraints and objectives. When the problem space is large enough to make optimization intractable, given the designer's limited processing power, the designer will "satisfice" or "find decisions that are good enough" using heuristic search [7]. Schön [9] named this view "Technical Rationality".

Reflection-in-Action is explicitly presented as an alternative to Technical Rationality [9]. Reflection-in-Action models design as a reflective conversation in which the designer alternates between framing (conceptualizing the problem), making moves (a move is a real or simulated action intended to improve the situation) and evaluating those moves [9]. Schön was not saying that Technical Rationality is wrong or ineffective; he was saying that, typically, constraints and objectives are not known so designers have to construct both problem and solution together. Later research showed that searching is not a good way to describe design thinking [33] and designers cannot reliably assess the quality of an on-paper solution candidate (cf. [5]).

In software engineering specifically, Technical Rationality is consistent with the view of projects having a design activity or phase that is separate from requirements, analysis, programming and testing. This design phase is sometimes associated with software architecture and seen as determining most or all of the software artifact's important properties. Reflection-in-Action is consistent with the idea that analyzing the situation and designing the system are a single, inextricable process. Moreover, numerous studies have shown that developers make design decisions while coding (cf. [34,35]), and refactoring has long been understood as a kind of design [36]. In other words, design pervades a software project with properties of the artifact determined throughout its development; for example:

- by salespeople selling a proposed solution
- by analysts during problem framing
- by negotiators during contract negotiation
- by a project manager or product owner during a planning meeting
- by a software developer while programming, refactoring or fixing bugs

This does not mean that projects cannot have architects or segments of time dedicated to high-level design. The key difference between Technical Rationality and Reflection-in-Action is whether design is seen as a loosely-coupled phase

within a broader development process or as a ubiquitous activity that occurs throughout development and is intertwined with analysis and programming.

Technical Rationality and Reflection-in-Action differ in their epistemic justification. Reflection-in-Action is rooted in pragmatism [9], which is a form of empiricism [37]. Schön cites extensive observations of professional designers to justify Reflection-in-Action. Extensions of Reflection-in-Action are similarly based on empirical studies (e.g. [14]). In contrast, Technical Rationality invokes The Intuition/Deduction Thesis by relying on rationalist justification. Simon cites no observational studies to justify the accuracy of Technical Rationality's propositions concerning how real designers think and behave. For example, Simon claims that "problem-solving systems and design procedures in the real world ... must *search* for appropriate assemblies" [7 original italics]. He provides no empirical evidence to support this claim, which was subsequently contradicted by empirical studies [38,39].

Simon also posits that designers exhibit "procedural rationality." "Behavior is procedurally rational when it is the outcome of appropriate deliberation" [40]; i.e., although they may not identify the optimal solution, they act rationally given their limited processing power. Again, no empirical evidence is provided to support the conjecture that designers are procedurally rational in practice and it is not consistent with empirical observations [34,35].

To be clear, Technical Rationality and Reflection-in-Action are scientific descriptions of how professionals act in situ. They are not methods. No one is arguing that Reflection-in-Action and Technical Rationality can be adopted at will, or that adopting either would improve performance. Schön's criticism of Technical Rationality is *people do not act this way in real life*, not *acting this way is ineffective*.

3.2. Model of human action

The two paradigms also embrace different models of human action. In the Rational Paradigm, it is widely assumed that human action is predominately plan-execution. This "Planning Model" of human action entails three propositions: 1) planning is a prerequisite to acting, 2) actors understand their actions and progress in terms of the plan, and 3) good planning is a primary determinant of success [41]. In the Empirical Paradigm, it is widely assumed that human action is predominately improvisation. This "Improvising Model" of human action posits that "the organization of situated action is an emergent property of moment-by-moment interactions between actors, and between actors and the environments of their action" while "plans are representations, or abstractions over action" [42]. In other words, the Planning Model assumes that plans are strong resources for guiding actions, while the Improvising Model assumes plans are weak resources for guiding actions. In the Planning Model, good planning must be important because we cannot act without a plan. Straying from the plan is often seen as deviant behavior. In the Improvising Model, planning is seen as less important because people are always improvising. Moreover, improvisation, workarounds, hacks and generally going off script are often benevolent, helpful or necessary.

Proponents of the Planning Model tend to appeal to common sense or other intuitive justification that human action is plan-centered (e.g. [43]), implicitly adopting The Intuition/Deduction Thesis. In contrast, proponents of the Improvising Model (e.g. [41,44]) tend to adopt The Empiricism Thesis by citing empirical studies and providing empirical observations to justify it. Empirical research demonstrates that human action is not simply plan-execution because individuals often deviate from detailed plans, planning is often tangential to action and some actions are not planned at all (see [41] for review). Designers will claim to be following a plan even while obviously deviating from it [45]. Like Schön [9] with Reflection-in-Action, Suchman [41,42] deconstructs a worldview that is intuitively appealing but contradicted by empirical observations.

Refuting the Planning Model is extremely controversial. Scholars tend to raise at least two objections:

1. They claim that in their experience, good planners are more successful than people who improvise.
2. They point to large, complicated systems in aerospace, defense, e-government, shipbuilding, etc. and claim that it is intuitively obvious that these systems could not arise without good planning.

Both of these objections appear reasonable but misunderstand the Planning and Improving Models. The first objection confuses The Planning and Improvising Models, which are scientific descriptions of human cognition, with *techniques*. Everyone makes plans at least some of the time. The debate here is about the extent to which actors focus their mind on the situation at hand or a pre-existing plan while acting. It is not about whether meticulously planning a project is more effective than just winging it.

The second objection conflates planning with prospective sensemaking. Sensemaking is the process of making sense of, or assigning meaning to, experiences [46]. Making sense of past experiences is called *retrospective sensemaking*. Exploring and interpreting an imagined future, including strategizing and planning (that is, the sensemaking of designers [47]) is called *prospective sensemaking*. In other words, '*how can you build complicated software without good planning?*' conflates planning with *preparing*. Planning is only one way of preparing, and much of the preparation in software projects, including writing stories and scenarios, is better understood as prospective sensemaking [48]. Again, the improvising model does not justify or promote cowboy coding or a 'just wing it' attitude.

In SE, the Planning Model underlies stage-gate or lifecycle views of the design process [49], planning and productivity in Agile and Heavyweight methods (below) and research on control (e.g. [50,51]). Meanwhile, the Improvising Model underlies

teleological views of the design processes [10,52], amethodical development (below) and research on emergence, complexity and paradox (e.g. [44,53,54]).

Additionally, Technical Rationality and many software development methods present design as a kind of planning; specifically, converting a requirements specification into a plan for constructing an artifact (cf. [55–57]). This again conflates planning with prospective sensemaking. Reflection-in-Action contrastingly views design as a kind of improvising in which a designer simultaneously refines a problem and a solution candidate (cf. [9,10,58]). A design project is therefore seen as a system of inquiry in which designers co-construct desiderata with stakeholders and generate solutions by building artifacts in the world rather than planning them in the mind.

3.3. View of success and requirements

In Software Engineering specifically, the Rational and Empirical Paradigms involve very different (and interconnected) views of success and requirements.

Research in the Rational Paradigm tends to model success based on the *Project Triangle* (also called the *Iron Triangle*). That is, success means delivering the agreed requirements (scope) within budget, on time and at an acceptable quality level [59]. Research in the Empirical Paradigm contrastingly tends to model success based on the system's impact on its stakeholders [60,61]. These views differ substantially in epistemic justification. The Project Triangle is not a scientific theory – its origins are unclear and no empirical studies support its appropriateness as a theory for understanding success. Users of the Project Triangle tend simply to invoke it without justification, or to just claim that it is obvious (i.e. invoking the Intuition-Deduction Hypothesis). Proponents of the stakeholder-impact view tend to conduct or review empirical studies supporting their models (e.g. [60,61]). They tend to highlight examples of systems that are widely considered successful despite being delivered late, over-budget or not to specification [62].

These differing models of success lead to differing views of software requirements. To understand the differences, we can distinguish between:

- a *desideratum*: a property of a real or imagined system that is wanted or needed by one or more system stakeholders [5]; and
- a *requirement*: a property that a real or imagined system must possess [18].

In other words, desiderata include every property of a system that one or more stakeholders wants or needs, and requirements are a subset of desiderata that are necessary for success. In Section 2, we saw how rationalists and empiricists differ in their views of what justifies a belief. Similarly, research in the Rational and Empirical Paradigms differ in their views of what justifies labeling a desideratum "necessary for success"; that is, a requirement.

For example, suppose that we are developing a payroll system. Further suppose that the client organization demands that the system should enforce strong passwords, and we add an equivalent term to the contract. Suppose no one disputes that strong passwords are a desideratum.

In the Rational Paradigm, success means delivering agreed functionality, so agreeing the functionality makes it necessary for success. No further justification is needed to designate something a requirement.

In the Empirical Paradigm, success is about the effects a system has on its environment. We might therefore argue that strong passwords are not necessary for success on at least two grounds:

1) Several good design alternatives exist, including two-factor authentication, biometrics and single sign-on. The system could therefore achieve its objectives and deliver intended benefits without strong passwords.
2) Many successful systems (including many banking systems) do not enforce strong passwords. We therefore have no empirical justification for presuming that lack of strong passwords would result in system failure.

Research in requirements engineering has attempted to acknowledge these kinds of objections by introducing "optional requirements," priority levels (e.g. [63]) and confidence levels (e.g. [64]). In the Empirical Paradigm, all but some of the top priority, highest-confidence desiderata would be viewed as preferences and conjectures.

Moreover, differing underlying philosophies lead to very different terminology and assumptions in the two paradigms. In the Rational Paradigm, analysts "elicit" requirements by interviewing stakeholders, or "discover" them by examining legacy systems and analyzing discourse [65]. The analyst then strives to produce an "unambiguous", "consistent", "complete", "feasible", "traceable" and "verifiable" requirements specification [66]. Terms including "elicit" and "discover" reveal the assumption that requirements exist in an observer-independent objective reality. This assumption pervades industry standards (e.g. [66]), software development methods (e.g. [67,68]) and model curricula (e.g. [69,70]).

Contrastingly, research in the Empirical Paradigm usually assumes that social reality is subjective. All knowledge, including requirements, are viewed as constructed rather than discovered [71]. Moreover, "one of the main themes that has emerged from behavioral decision research during the past three decades is the view that people's preferences are often constructed in the process of elicitation" ([72], p. i). Requirements engineering is therefore a creative process [73] where analysts and stakeholders collaboratively make sense of the current situation and construct a shared mental model of a possible system [74]. This does not mean that the Empirical Paradigm interdicts requirements engineering or even big-

Table 1
Examples of process models and theories, by paradigm.

Empirical Paradigm	Rational Paradigm
The Selfconscious Process [82]	The Waterfall Model [8]
The Theory of Persistence and Change in Software Development [83]	The Spiral Model [86]
The Play-Test Boomerang [84]	The Function–Behavior–Structure Framework [55]
The Theory of Distances in Software Development [85]	The V-Model [87]
Sensemaking–Coevolution–Implementation Theory [10]	The Pull Model of Development [88]
	The solution progression in the Essence standard [89]

design-up-front. It means that most results of requirements engineering are viewed as desiderata, ideas or conjectures, rather than requirements per se.

In the Rational Paradigm, the analyst is seen as discovering requirements in an objective reality. The results of "elicitation" are therefore readily accepted as requirements. In the Empirical Paradigm, contrastingly, the analyst is seen as co-constructing preferences with stakeholders and the success of a design candidate is discernible only by constructing it (cf. [5]). The results of stakeholder interviews, focus groups, up-front analyses and contract negotiations are therefore mainly viewed as desiderata and conjecture. For a desideratum, *d*, to be a requirement, it must meet two conditions:

1. *d* is necessary for the system to succeed, that is, any system without *d* will fail; and
2. our belief in condition 1 is justified, that is, we *know* that any system without *d* will fail. [75]

Practically speaking, many things *presented* as requirements do not meet these conditions, so the empiricist rejects most ostensible requirements as unjustified desiderata (cf. [75–77]). This reinforces the view that designing is more like improvisation than planning, (cf. [39,78]) and that problem framing and solution generation are simultaneous and interconnected [9]. The Rationalist, in contrast, can draw neat deductive lines between given problems, clear, elicited requirements and a largely independent, planning-oriented design phase.

3.4. Models of the development process

Software developers engage in myriad activities including abstracting, associating, decision making, information gathering, modeling, searching, selecting and synthesizing [79]. Many researchers have organized these low-level activities into higher-level categories to manage cognitive load and to understand development. This produced numerous process models and theories. For the purposes of this paper, a *process model* "is an abstract description of an actual or proposed process that represents selected process elements that are considered important to the purpose of the model and can be enacted by a human or machine" [80]. Meanwhile, a *process theory* is an explanation of how and why an entity changes and develops [81].

We can categorize process models and theories according to which paradigm better reflects their assumptions (Table 1). Unsurprisingly, those categorized under the Empirical Paradigm in Table 1 are all predominately generated from or justified by empirical observation [10,82–85]. Contrastingly, those categorized under the Rational Paradigm are all predominately generated from and justified by argumentation, intuition, deduction and anecdotal evidence [8,55,86–89].

Beyond their epistemology, these models differ in at least three ways:

1. The Empirical Paradigm models are predominately descriptive or explanatory, as is obvious from their respective texts. The Rational Paradigm models are all prescriptive. This is obvious in the texts presenting Spiral, V, Pull and Essence. Royce clearly argued that the forward-only Waterfall Model was ineffective; however, many others championed Waterfall, and the more iterative model Royce proposed is also part of the Rational Paradigm. Gero was less obviously prescribing but later analysis shows how his model was simultaneously descriptive and prescriptive [90]. Of course, the texts proposing the Empirical Paradigm models sometimes give suggestions, and the texts proposing the Rational Paradigm models sometimes describe and explain – it is a matter of degree.
2. Process models and theories come in four basic varieties [91]: lifecycle (sequence of phases), evolutionary (evolving populations), dialectic (conflict between several entities) and teleological (goal-oriented agent choosing actions). All of the Rational models except The Function-Behavior-Structure Framework are presented as lifecycles. None of the Empirical models are presented as lifecycles – the Theory of Distances and Sabherwal and Newman's theory are dialectic theories; the rest are teleological. This is striking because lifecycle approaches have been criticized as too inflexible for theorizing human behavior [92].
3. The two categories of models use substantially different terminology. The Rational models tend to organize development activities into minor variations of *requirements*, *analysis*, *design*, *coding* and *testing* – here called *Royce's Taxonomy* because of their similarity to the Waterfall Model. All of the Empirical models deviate substantially from Royce's Taxonomy.

Royce's Taxonomy – not any particular sequence – has been implicitly co-opted as the dominant software development process theory [5]. That is, many research articles, textbooks and standards assume:

1) Virtually all software development activities can be divided into a small number of coherent, loosely-coupled categories.
2) The categories are typically the same, regardless of the system under construction, project environment or who is doing the development.
3) The categories approximate Royce's Taxonomy.

For example, Fitzgerald [93] states that "in conventional software development, the development lifecycle in its most generic form comprises four broad phases: planning, analysis, design, and implementation" (p. 3). Similarly, Ewusi-Mensah [94] says that "regardless of the particular process model ... every software project will feature: (1) the requirements-definition and functional-specification phase; (2) the design phase; ... (3) the implementation; ... and (4) the installation, operation, and maintenance phase" (p. 51). Additionally, Laudon et al. [95] state that "systems development ... consist[s] of systems analysis, systems design, programming, testing, conversion and production and maintenance ... which usually take place in sequential order". Moreover, Royce's Taxonomy is evident in the IEEE Guide to the Software Engineering Body of Knowledge [96] and implicitly adopted by numerous standards (e.g. [66,89,97]).

Royce's Taxonomy is so ingrained as the dominant paradigm that it may be difficult to imagine a fundamentally different classification. However, good classification systems organize similar instances and help us make useful inferences [98]. Like a good system decomposition, a process model or theory should organize software development activities into categories that have high cohesion (activities within a category are highly related) and loose coupling (activities in different categories are loosely related) [99].

Royce's Taxonomy is a problematic classification because it does not organize like with like. Consider, for example, the design phase. Some design decisions are made by "analysts" during what appears to be "requirements elicitation", while others are made by "designers" during a "design meeting", while others are made by "programmers" while "coding" or even "testing." This means the "design" category exhibits neither high cohesion nor loose coupling. Similarly, consider the "testing" phase. Some kinds of testing are often done by "programmers" during the ostensible "coding" phase (e.g. static code analysis, fixing compilation errors) while others often done by "analysts" during what appears to be "requirements elicitation" (e.g. acceptance testing). Unit testing, meanwhile, includes designing and coding the unit tests.

Because Royce's Taxonomy does not organize like activities together, it does not support many useful inferences. For example, suppose we know that an activity, A, is a member of the "testing" class. What else can we infer about A? Because testing includes such diverse activities, we cannot infer who is involved (programmers? analysts? users?), what kind of skills are needed (interpersonal? technical?) or when the activity occurs (early in the project? late? throughout?).

It should not be surprising, then, that models and theories based on empirical observation suggest terminology and organization vastly different from Royce's Taxonomy. For example, Sabherwal and Newman's [83] dialectical theory is predominately concerned with the relationship between *persistence* and *change* rather than *analysis* and *design* or *coding* and *testing*. The Play-Test Boomerang [84] posits that digital game development is primarily driven by continuous playtesting. *Playtesting* refers to the process of a developer playing a game to evaluate qualitative characteristics including immersion, balance and difficulty [100]. Playtesting differs fundamentally from testing in Royce's Taxonomy or even Test Driven Development [101] in emphasizing the dynamics and aesthetics of the system rather than its performance, features and bugs.

Meanwhile, Sensemaking–Coevolution–Implementation Theory [10,102] explicitly reorganizes development activities to improve cohesion and coupling. *Sensemaking* combines domain or environment-focused analysis (e.g. developing personas [103]) with user-focused testing (e.g. acceptance testing). *Implementation* combines low-level design decisions (e.g. between a for loop and a while loop) with coding and technical testing (e.g. unit, integration, system, performance). *Coevolution* combines high level design decisions (e.g. architectural patterns) with solution-focused analyses (e.g. analyzing design candidates).

In summary, the process models and theories associated with the Rational Paradigm:

- are largely justified using rational argument, intuition, deduction and anecdotal evidence;
- use categories, phases or activities similar to that of the Waterfall Model, but different sequences;
- are predominately presented as lifecycle models; and
- are often prescriptive.

Contrastingly, the models and theories associated with the Empirical Paradigm:

- are largely justified using empirical observation;
- use more diverse activities, processes and constructs;
- are presented as teleological or dialectic models; and
- are predominately descriptive or explanatory.

3.5. Software development methods

A *System Development Method*, or simply a *method*, denotes a collection of interconnected prescriptions for building systems effectively and connotes orderly, predictable approaches that apply in diverse settings [104]. Methods are often divided into two groups (cf. [16]) – *Agile Methods* including Scrum [105], Extreme Programming [88] and Lean [106] versus *Heavyweight Methods* including The Rational Unified Process [107], PRINCE2 [108] and the Waterfall Model [8]. Superficially, Agile methods seem more consistent with the Empirical Paradigm while Heavyweight methods seem more consistent with the Rational Paradigm. More fundamentally, however, the situation is messy.

3.5.1. Epistemology and methods

Regarding epistemology, the effectiveness of methods is typically justified using a combination of anecdotal evidence, argument, intuition and deduction. As explained in Section 2.3, using anecdotal evidence to support unobservable properties such as effectiveness appeals to Rationalism. In engineering research, creators of new artifacts, including methods, are normally expected to demonstrate that the proposed artifact outperforms an existing baseline artifact [109–111]. None of the books that propose any of the methods discussed in this article present any such empirical demonstration. Indeed, I am not aware of any credible research demonstrating that any method outperforms any other method on any measurable dimension (cf. [58,112]). Method proponents do sometimes cite empirical studies that support some aspect of the method, but not the method itself. The Agile community, in particular, has proposed copious ostensibly Agile practices without ever demonstrating that any of these practices have ever made any particular software team more agile. Instead, they subtly re-define agility as the extent to which the team adheres to their recommendations (a circular argument) rather than the extent to which a practice makes a team more nimble and responsive (cf. [113,114]). In summary, although the methods literature does not explicitly embrace rationalism, its implicitly invokes rationalism by relying on intuitive argument and anecdotal evidence.

3.5.2. Model of design and methods

Heavyweight methods tend to treat design as a phase (e.g. Waterfall) or as a stream of work that is loosely coupled with analysis and software construction (e.g. RUP), consistent with technical rationality. Agile methods *sometimes* present analysis and design as intertwined, consistent with reflection-in-action, but this is not always clear. Both Scrum and XP begin each sprint with a time-boxed planning meeting, suggesting a clear distinction between design and development. Similarly, the process of adding to the backlog (problem framing) is often presented as largely disconnected from the process of determining how to implement backlog items (problem solving). This is further complicated by the Agile literature rarely using terms like problem framing, design concept generation or coevolution. In fact, seminal books on XP [88], Scrum [105,115] and Lean [106] do not explain much about how software teams either generate and evaluate high-level design alternatives or make low-level design decisions. In summary, while heavyweight methods typically embrace Technical Rationality, Agile methods do not share a clear or consistent model of design.

3.5.3. Model of human action and methods

The relationship between the Improvising/Planning conflict and the Agile/Heavyweight conflict is also complicated. Heavyweight methods are clearly consistent with the Planning Model, but Agile methods are not clearly consistent with either model. On the one hand:

- The Agile Manifesto explicitly values “responding to change over following a plan” [116].
- Agile methods obviously do not involve as much planning as Heavyweight methods.

On the other hand:

- Agile methods recommend organizing development into cycles of planning (e.g. planning meeting), acting (e.g. the sprint) and reflecting (e.g. review meeting).
- Agile methods prescribe various techniques to help teams understand their progress in terms of the plan, including prioritized burndown charts and daily meetings.

It is tempting, then, to assume that the Agile movement occupies a middle ground between the two views. This would be misleading, however. The planning–improvising conflict is not about whether we make plans; no one is disputing that people make plans. The planning–improvising conflict is about whether the programmer coding a method, researching a library or reviewing test results is predominately executing a plan or spontaneously reacting to a dynamic situation. When stated this way, my impression of the Agile literature is that it is more consistent with viewing developers as executing (short-term) plans. However, the Agile literature rarely addresses cognitive processes directly, so this interpretation is debatable.

Moreover, the Planning/Improvising conflict is not a spectrum with a middle to occupy. The Planning and Improvising Models make conflicting predictions about the cognitive processes underlying human action. Either developer cognition is predominately situated improvisation or plan-execution; we have no theoretical basis to hypothesize their combination. Assuming the truth lies in between commits the middle ground fallacy.

3.5.4. Summary

In summary, the methods literature is broadly consistent with the Rational Paradigm because it uses Rationalist epistemological justifications. Most heavyweight methods are clearly consistent with the Rational Paradigm. Some Agile methods, however, present internally inconsistent assumptions that do not fit cleanly with either paradigm.

3.6. Practical aim: description vs. prescription

Texts in the Empirical Paradigm tend to *describe* and *explain* how software professionals act. Much of the Empirical Paradigm comprises qualitative research, which generates concepts, themes, taxonomies or process theories (see Section 3.4). When texts in the Empirical Paradigm do make recommendations, they tend to concern individual practices and tools rather than comprehensive methods. Unlike methods, many studies empirically compare specific practices on measurable dimensions. For example, Hannay et al. recommend "pair programming either when task complexity is low and time is of the essence, or when task complexity is high and correctness is important" [117] based on a meta-analysis of pair programming experiments. Hannay et al. invoke The Empiricism Thesis by justifying their suggestions based on empirical studies.

Texts in the Rational Paradigm tend to *prescribe* both specific methods and methodicalness in general. Although not all software development methods clearly embrace the Rational Paradigm, the Rational Paradigm clearly embraces methods in a way that the Empirical Paradigm does not. The Rational Paradigm includes many texts encouraging developers to act methodically and rationally (e.g. [32,118–120]). Even when these texts recognize that perfectly rational development is impossible, they maintain that rationality and methodicalness are desirable and should be maximized [119]. A related claim is that most or all projects follow the same sequences of phases or lifecycle [8,67,68] – see above. These texts often make assumptions similar to those of Technical Rationality; for instance, Royce [8], Kruchten [107] and Jacobson et al. [121] all present problem solving and problem framing as loosely coupled.

In contrast, the Empirical Paradigm tends to reject universality and accept that a more systematic approach may be better in some contexts (cf. [122]) while a less systematic approach is better in others (cf. [44]). *Amethodical development* "implies management and orchestration of systems development without a predefined sequence, control, rationality, or claims to universality" [104] and connotes activity that is unique and unpredictable but not chaotic. The Empirical Paradigm respects the reality that methods are neither extensively nor effectively used [123–126], methods cannot be used directly [127], developers waste time faking method use [128,129], improvisation is the basis of much development activity [44,52,130,131] and "amethodical development might be the normal way in which the building of these systems actually occurs in reality" [104]. Like Schön [9] and Suchman [41], Truex et al. [104] identify an intuitively appealing narrative, point out that proponents of this narrative have not provided empirical evidence to support their claims and enumerate a host of studies that support a different worldview. To be clear, these scholars do not argue that methods and methodicalness are *always* harmful, or that developers *never* apply methods. They simply point out that empirical research supports neither method effectiveness nor widespread use. Like plans, they see methods as weak resources for informing and explaining human behavior.

In summary, since the Rational Paradigm promotes rationality and methodicalness, it is consistent with prescribing specific methods of developing software. In contrast, the Empirical Paradigm questions the usefulness, effectiveness, applicability and universality of methods. It questions whether and in what circumstances rationality and methodicalness are possible and desirable. To the extent that it makes prescriptions, it prescribes individual practices and tools based on empirical evidence of their effectiveness. The Rational and Empirical Paradigms fundamentally disagree on *when methodicalness is desirable and whether methods are useful*. This has nothing to do with the debate between Agile and Heavyweight methods, which concerns *which methods are better* (cf. [16]).

3.7. Defining the Rational and Empirical Paradigms

Kuhn defined a paradigm as a "constellation of beliefs, values, techniques, and so on shared by the members of a given community" [4]. The preceding discussion demonstrates that the scholarly literature on design and software development manifests two very different constellations of beliefs, values and techniques. One group predominately uses empirical research to justify descriptions of how software is developed. The other predominately uses rational argumentation, intuition, deduction and anecdotal evidence to justify prescriptions on how software should be developed. The Empirical and Rational Paradigms are the worldviews underlying each of these groups. The worldviews themselves have many interconnected components, summarized in Table 2. More formally, the two paradigms can be defined as follows.

The Rational Design Paradigm: *the constellation of beliefs, values and techniques* (*including Technical Rationality, the Planning Model of human action and a Project-Triangle view of success*), *which underlies a mutually-reinforcing, design-related stream of research that is justified mainly by rational argumentation, intuition, deduction and anecdotal evidence*.

The Empirical Design Paradigm: *the constellation of beliefs, values and techniques* (*including Reflection-in-Action, the Improvising Model of human action and a stakeholder-impacts view of success*), *which underlies a mutually-reinforcing, design-related stream of research that is justified mainly by empirical observation*.

Table 2
Summary of the Rational and Empirical Paradigms.

Dimension	Rational Paradigm	Empirical Paradigm
Epistemology	Rationalism	Empiricism
Model of Design	Technical Rationality	Reflection-in-Action
Model of Action	Planning Model	Improvising Model
View of Success	Project Triangle	Stakeholder Impacts
View of Desiderata	Mostly Requirements	Mostly Conjecture
Model of the Development Process	Mostly Lifecycle Process Models and Methods	Dialectic or Teleological Process Theories
Associated Methods	All Heavyweight; some Agile	Amethodical
Practical Aim	Prescription	Description

Table 3
Common positions of the Rational and Empirical Paradigms.

Rational Paradigm position	Empirical Paradigm position
Developers use methods [88,132]	Developers rarely use methods as intended or at all [123–126]
Functional requirements exist, in an objective reality, waiting for discovery [133]	System requirements are an illusion motivated by our unwillingness to recognize epistemic uncertainty [75]
Development teams elicit requirements from users [65]	Development teams make sense of a problematic context [134,135]
Success means delivering the required scope within the established budget and schedule [136]	The Project Triangle is oversimplified [59]. Success is "a multidimensional variable comprising project efficiency, artifact quality, market performance and stakeholder impacts over time" [61]
More clear, structured, well-defined problems are easier to solve [32]	Clarifying and structuring problems decreases design performance [39,76,77]
Design is a phase or part of development, temporally or conceptually situated between analysis and coding [8,55]	Design encompasses the entire development process from initiation to maintenance [19,58]
Development solves given problems [7]	There is no "the problem", only a context that some actors view as problematic [15]

Note: the references cited in this table are examples of texts that adopt (rather than provide evidence for) each related position.

4. Discussion

The elucidation of the two paradigms, above, raises three fundamental questions: 1) how do the differing worldviews affect the way scholars conceptualize software development; 2) which paradigm is more useful for understanding and improving software development; and 3) how can the Rational Paradigm survive in a research community that is increasingly concerned with empirical data? This section addresses each of these questions.

4.1. Paradigms as narratives

The Empirical and Rational Paradigms embrace conflicting metanarratives (Box 1). These metanarratives include numerous inconsistent positions (Table 3).

At the heart of the Rational Paradigm is a story about how software is made. Analysts use uncontroversial goals to elicit clear, comprehensive, agreed requirements. Designers search a conceptual solution space for satisfactory design candidates. Developers select an appropriate development method and use it to construct software as specified in design documents. The development process is carefully planned as a series of activities or phases with milestones. Software professionals not only execute the plan but also understand and evaluate their progress in terms of the plan. Unexpected events trigger re-planning and straying from the plan is viewed as deviant. While acknowledging the impossibility of perfect rationality, professionals are encouraged to be as structured, methodical and rational as they can. Researchers and professionals use lifecycle models and development methods to prescribe actions.

This view of development is broadly appealing for many reasons. It was developed by widely respected, influential academics. It is relatively simple and easy to explain to students. It legitimizes software development by presenting it like engineering. It corresponds well to existing research in artificial intelligence and utility theory (e.g. [137]). Moreover, its language and assumptions are consistent with many methods and process models including PRINCE2 [108] and Capability Maturity Model Integration [138]. It is compatible with contemporary approaches to governance, financial management and outsourcing [5]. Moreover, being the status quo, it benefits from a variety of psychological biases that inhibit change [139, 140].

Unsurprisingly then, The Rational Paradigm is the dominant paradigm in SE (and information systems, project management and several other fields, but that is beyond the scope of this paper) as seen in textbooks [e.g. 95], model curricula [70,141], standards (e.g. [66,89,97]), official bodies of knowledge (e.g. [96]), Wikipedia articles (e.g. the ubiquitous Software Development Process Template), guidance for design science research [110], the elevation of Royce's Taxonomy to a de facto process theory (discussed above) and even popular conceptions of the scientific process (e.g. [142–144]).

As early as the 1970s, however, researchers began pointing out that this story is inconsistent with their observations:

1976: *An organization does what it does because of plans, intentional selection of means that get the organization to agree upon goals, and all of this is accomplished by such rationalized procedures as cost-benefit analyses, division of labor, specified areas of discretion, authority invested in the office, job descriptions, and a consistent evaluation and reward system. The only problem with that portrait is that it is rare in nature. People in organizations, including educational organizations, find themselves hard pressed either to find actual instances of those rational practices or to find rationalized practices whose outcomes have been as beneficent as predicted, or to feel that those rational occasions explain much of what goes on within the organization. Parts of some organizations are heavily rationalized but many parts also prove intractable to analysis through rational assumptions.* [145]

1983: *According to the model of Technical Rationality – the view of professional knowledge which has most powerfully shaped both our thinking about the professions and the institutional relations of research, education, and practice – professional activity consists in instrumental problem-solving made rigorous by the application of scientific theory and technique ... Technical rationality depends on agreement about ends... but when ends are confused and conflicting, there is as yet no "problem" to solve... Similarly, when there are conflicting paradigms of professional practice... there is no clearly established context for the use of technique... and when practitioners do resolve conflicting role frames, it is through a kind of inquiry which falls outside the model of Technical Rationality*. [9]

1987: *Because the circumstances of our actions are never fully anticipated and are continuously changing around us ... our actions, while systematic, are never planned in the strong sense that cognitive science would have it. Rather, plans are best viewed as a weak resource for what is primarily ad hoc activity. It is only when we are pressed to account for the rationality of our actions, given the biases of European culture, that we invoke the guidance of a plan. Stated in advance, plans are necessarily vague, insofar as they must accommodate the unforeseeable contingencies of particular situations. Reconstructed in retrospect, plans systematically filter out precisely the particularity of detail that characterizes situated actions, in favor of those aspects of the actions that can be seen to accord with the plan.* [42]

1992: Conventional wisdom about problem-solving seems often to be contradicted by the behavior of expert designers. But designing has many differences from conventional problem-solving [146]*. For example, in designing, "the solution" does not arise directly from "the problem"; the designer's attention oscillates, or commutes, between the two, and an understanding of both gradually develops.* [12]

These quotations all make the same argument about different aspects of the Rational Paradigm: it appears reasonable but is contradicted by empirical observations. Their work, together with other research reviewed in this paper, suggests a very different metanarrative.

Projects have diverse stakeholders with different values and goals, which may be unclear. Software professionals work with stakeholders to co-construct desiderata, very few of which are requirements per se. Development teams simultaneously develop software artifacts and explore the problematic context, which is complex, unpredictable and unstable. Though professionals make plans, they are a weak resource for predominately improvised behavior. When professionals (rarely) use a method or process model, they do not use it as intended by its creators. Researchers use process theories and models to describe and explain development phenomena, and occasionally recommend specific practices based on empirical evidence.

Of course, an individual project, team, organization or context may resemble the Rational Narrative, resemble the Empirical Narrative, combine elements of both or be radically different from either. The Rational and Empirical Paradigms refer to worldviews underlying bodies of research, not collections of organizations, teams or projects.

4.2. Paradigms and puzzle solving

Kuhn [4] argues that the dominant criterion of paradigm quality is usefulness for solving scientific puzzles; that is, better paradigms are more useful *to scientists* for understanding, describing, explaining and predicting phenomena. Paradigms are not necessarily useful for or usable by non-scientists. The Empirical and Rational Paradigms may therefore be evaluated based on their usefulness for solving scientific puzzles including:

- How *do* and how *should* developers build systems?
- What causes software projects to succeed or fail?
- How do specific tools, techniques, behaviors and conditions affect software projects?

Technical Rationality and Reflection-in-Action are used to understand design activity. In different circumstances, each may be used to describe designing [1]. The heart of Schön's argument for Reflection-in-Action is that when the designer frames as well as solves a problem, "Technical Rationality appears as radically incomplete" [9]. In other words, Technical Rationality is not useful for understanding real-world design activity when design problems or goals are ambiguous, contested, or simply unknown (cf. [15]). In such situations, designers frame problems and generate solution candidates simultaneously [9,11,12] – behavior incommensurate with Technical Rationality.

Similarly, the Planning and Improvising Models of action are used to understand behavior. Suchman [42] concludes that adopting the Planning Model obscures improvisation and the effects of context on interpreting action. It predisposes the investigator toward simpler, post hoc rationalizations of human behavior [53]. For instance, the planning model obscures the empirical question of whether planning is helping or hindering a given situation. The Improvising Model is more useful because it facilitates more nuanced, less biased interpretations of actions. In software development specifically, empirical evidence of the primacy of improvisation (cf. [44,52,131,147]) suggests that the Planning Model would be especially misleading.

While methods may be practically useful for developers and managers [122,148,149], are they useful for solving scientific puzzles? Clearly, if one is studying a team that is heavily influenced by a particular method, understanding the method may help us understand the team. However, substantial evidence suggests that methods are neither effectively nor extensively used [123–126,128] and when they are used, they are not used as intended [125,127,150]. Rather, "methodologies are treated primarily as a necessary fiction to present an image of control or to provide a symbolic status, and are too mechanistic to be of much use in the detailed, day-to-day organization of systems developers' activities" [151]. Analyzing developer behavior through a methods lens is therefore likely to encourage oversimplified and overrationalized views of development, which obscure complexity, paradoxes and innate tensions [53]. Analyzing success and failure through a methods lens is similarly likely to fixate researchers on methodicalness [104] and fictitious method features (e.g. method compliance) rather than team features (e.g. team cohesion) [152].

Similarly, while Rational process models and theories may be useful for developers and managers in some situations [153], they may be less useful for solving scientific puzzles. All of the Rational process models and theories identified above are prescriptive. As Vermaas and Dorst [90] explain: "Descriptive models ... apply equally to successful design cases and to less successful ones. A prescriptive model can thus clearly not be descriptive as well since then less successful design cases conform also to the model" (p. 152). Analyzing development behavior, progression or outcomes through Rational process models therefore encourages researchers to ignore, downplay or perceive as deviant non-prescribed behaviors regardless of effectiveness [154]. It may also encourage oversimplifying tightly interconnected activities into discrete phases, simultaneous activities as sequential phases, or complex combinations of progression and regression as steady improvement.

For example, suppose we interview some developers. Using the concepts and assumptions of the Rational Paradigm, we might ask "What is the problem you are attempting to solve?" This a loaded question – it presupposes a single, clear, agreed aim, which encourages the interviewee to name a potential problem [155]. Using the concepts and assumptions of the Empirical Paradigm, in contrast, we might ask "Who are the stakeholders in this project?" and "To what extent do they agree on its aims and objectives?" By not presupposing agreement, the framing of these questions encourages the participant to reflect on the epistemic status of potential problem statements.

In summary, by focusing on prescription, the Rational Paradigm encourages researchers to oversimplify reality and focus on problems, requirements, plans, methods and phases that either do not exist or are tangential to what developers actually do. By focusing on explanation and description and emphasizing complexity, emergence and improvisation, the Empirical Paradigm encourages researchers to face real-world complexity and paradox. The Empirical Paradigm is therefore more useful for investigating scientific puzzles such as how developers work and why some systems fail.

4.3. To what extent has SE embraced empiricism?

It is extremely important not to confuse the Rational and Empirical Design Paradigms with Rationalist and Empiricist epistemologies. The Rational and Empirical Design Paradigms are different views of how software is designed and built. Rationalism and Empiricism are different views of where knowledge comes from. This paper argues that the Rational Paradigm is the dominant view of design in the SE community, not that Rationalism is the dominant philosophy. This section explores how ideas justified by Rationalism can survive in a field increasingly committed to empirical research.

Some reputable journals (e.g. *Requirements Engineering*) and conferences (e.g. *Fundamental Approaches to Software Engineering*) still publish studies with little empirical evaluation, and some research communities (e.g. programming languages) continue to accept anecdotal evidence in technical research [156]. However, the SE research community has obviously made great progress in methodological and scientific rigor. Many top outlets including the *International Conference on Software Engineering* and *IEEE Transactions on Software Engineering* are increasingly skeptical of papers without empirical data [157]. We have good journals (e.g. Empirical Software Engineering) and conferences (e.g. Empirical Software Engineering and Measurement) dedicated to empirical research. Additionally, empirical guidelines have been published for several types of studies (e.g. [158–160]).

Yet, many SE studies are only superficially empirical. As Ko et al. [161] explain: "Empirical studies, often in the form of controlled experiments, have been widely adopted in software engineering research as a way to evaluate the merits of new software engineering tools. However, controlled experiments involving human participants actually using new tools are still rare, and when they are conducted, some have serious validity concerns" (p. 110).

Suppose we develop a tool that generates examples from UML models to help users understand UML syntax and semantics. One good way to evaluate the tool is using a controlled experiment, in which some potential users are randomized into two groups. Individuals in the control group study some UML models and answer some questions. Individuals in the treatment group study some UML models with the help of the tool, and answer some questions. We hypothesize that the treatment group will answer more questions correctly than the control group. This directly addresses the dependent variable of interest: user understanding.

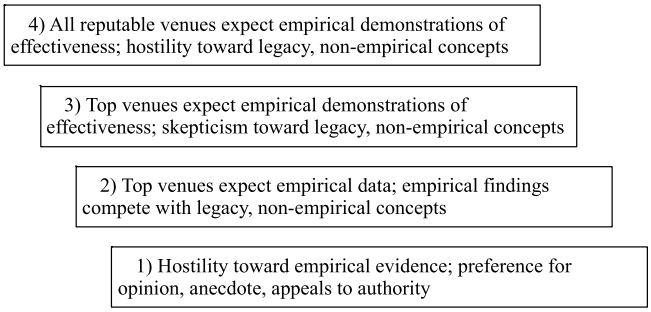

Fig. 1. Informal scale of empirical commitment.

Many SE studies avoid this kind of direct evaluation. Common strategies include the following.

- The researchers report technical performance data (e.g. number of semantically correct examples generated from a standard corpus of UML models; processing times). This simply describes the tool. The dependent variable of interest is not measured.
- The researchers apply the tool – themselves – to some real-life context and show that it generates reasonable examples. This demonstrates that the tool *can be used by expert researchers*, not that it improves understanding in non-expert professionals.
- The researchers ask some users to try the tool and give their opinion of its usefulness, usability and so on. This does not convincingly demonstrate anything because social desirability bias will inflate positive responses.

These strategies appear empirical in that they present data, but the data does not address the practical effectiveness or ostensible purposes of proposed artifacts. They are analogous to medical researchers testing drugs by asking patients whether they like them; checking whether they can be swallowed or evaluating their performance in a Petri dish. In other words, assessing artifacts in these ways is not wrong but *insufficient*.

Moreover, empiricism is not only about what kind of studies are published but also about what happens when empirical findings disagree with non-empirical concepts, models, anecdotes and the opinions of gurus.

SE professionals and researchers often appear to reject empirical evidence when it contradicts their beliefs. Software "practitioners seem to discount empirical evidence in favor of local opinion" [162]. Devanbu et al. [163] found that developers often hold strong opinions "for ideological reasons (or self-interest), even in the face of evidence to the contrary;" consequently, "practitioner beliefs can be inconsistent with project evidence". Meanwhile, Sharp et al.'s research on software quality management systems "revealed how practitioners had a strong sense of belonging to communities ... where the individualist and his (rarely her) opinions are highly valued, whether or not they are supported by evidence;" in other words, "empirical data was largely ignored in favor of position and opinion" [164].

Meanwhile, peer review is biased against new ideas [165] and "Reviewers [are] strongly biased against manuscripts which [report] results contrary to their theoretical perspective" [166]. In my limited experience as an author, reviewer, PC member and editor, rigorous empirical research is often criticized or rejected for contradicting concepts for which there was never any empirical evidence. This is crucial because the fundamental difference between science and pseudoscience is that science settles disputes using empirical evidence while pseudoscience suppresses contradictory findings [167]. Empirical findings that contradict widely-held, nonempirical beliefs therefore provide a kind of litmus test for a scientific community: insofar as empirical findings win out, SE is an applied science; insofar as anecdotes and nonempirical concepts win out, SE is pseudoscience and quackery. However, we have no quantitative estimates of how often empirical findings are ignored or how deleterious this is to practice.

Based on the preceding discussion, we can propose a rough scale to illustrate the possible range of embracing empiricism (Fig. 1).

At level one, we have pseudoscientific communities such as naturopathic medicine: evidence is irrelevant; the treatment is safe because it is "natural" and it works because I say it works for my patients and me. SE has clearly surpassed level one. At the opposite end, level four, we have mature empirical fields like medicine. In medicine, empirical data is required by all of the reputable journals, not just *The BMJ* and *The Lancet*. Moreover, studies have to demonstrate that artifacts actually achieve their goals; it is not enough to give some people a drug and ask them if they like it, or believe that it helped. Legacy, non-empirical concepts (e.g. phrenology) are viewed not neutrally but as threats to the credibility of the medical establishment. SE is clearly not at level four.

I think SE is at level two. Most top venues expect empirical data; however, that data often does not directly address effectiveness. Empirical findings and rigorous studies compete with non-empirical concepts and anecdotal evidence. For example, some reviews of a recent paper on software development waste [168] criticized it for its limited contribution over previous work [169], even though the previous work was based entirely on anecdotal evidence and the new paper was based on a rigorous empirical study. Meanwhile, many specialist and second-tier venues do not require empirical data at all.

While it seems esoteric, this discussion is critical for the future and impact of SE research. If the anecdotes of software gurus are perceived as more credible than the results of empirical studies, SE research is sabotaged from the outset. If no practical amount of empirical evidence is sufficient to abandon a non-empirical concept, how can past mistakes be corrected? If whenever evidence contradicts opinion, we ignore the evidence and retain the opinion, why bother collecting evidence at all? If software engineering aspires to be more science than religion, it is not enough to demand empirical evaluation for new research; we have to abandon nonempirical legacy artifacts.

Moreover, this position remains tenuous. There is no irresistible, invisible force driving scientific communities towards empiricism. The plethora of non-empirical articles and untested artifacts, combined with the tendency to reject findings that contradict personal experience and the opinions of gurus, continues to threaten our community's credibility.

In summary, SE reviewers increasingly expect empirical data. However, empirical data that does not directly address key dependent variables is still accepted. Moreover, empirical research is still criticized and rejected for disputing nonempirical legacy concepts. This allows the Rational Design Paradigm to survive.

5. Conclusion

This paper explores the crucial but opaque conflict between two contradictory views of how software is and should be developed. It makes three main contributions. The paper first reviews the philosophical conflict between Rationalism and Empiricism. It then demonstrates that the software engineering and interdisciplinary design discourse is bisected by two worldviews, one justified by Empiricism and the other justified by Rationalism. Finally, it explores the fundamental differences between these worldviews, leading to clear definitions of the Rational and Empirical Design Paradigms.

Briefly, research in the Rational Paradigm views designers as methodically searching for satisfactory solutions to given problems while following plans and using methods. Research on the Empirical Paradigm contrastingly views designers as simultaneously framing a problem from an ambiguous context and generating solutions through improvisation, while plans and methods are at best weak resources for informing and understanding their behavior.

This section describes the practical implications of these contributions, the limitations of the conceptual inquiry presented here and some potential avenues for future research.

5.1. Implications

The Rational Paradigm is like a myth about how software is created. In this myth, software professionals receive clear goals, carefully plan a project, elicit requirements, search for a satisfactory design candidate, and then use software development methods to construct the agreed functionality on schedule and within budget. This myth is evident in many official standards (e.g. the Software Engineering Body of Knowledge [96]), in Software Engineering education (cf. [170]) and in diverse scholarly work as reviewed throughout this article.

This article not only reviews the sizeable body of empirical research that contradicts the Rational Paradigm but also synthesizes a more realistic story about how software is created. Software professionals work with stakeholders to co-construct goals, ideas and preferences. They generate design candidates by alternating between reframing the problem, imagining fledgling solutions and building features. Plans and methods have little impact on professionals' moment-to-moment, improvised actions. The product is successful when it delivers benefits for stakeholders.

Changing the story researchers tell themselves and each other about how software is created has numerous implications for research, practice and education:

1) Much research involves developing new and improved development methods, tools, models, standards and techniques. Researchers who are unwittingly immersed in the Rational Paradigm may create artifacts based on unstated Rational-Paradigm assumptions, limiting their applicability and usefulness. For instance, the project management framework PRINCE2 prescribes that the project board (who set project goals) should not be the same people as project team (who design the system [108]). This is based on the Rationalist assumption that problems are given, and inhibits design coevolution.
2) Having two paradigms in the same academic community causes miscommunication [4], which undermines consensus and hinders scientific progress [171]. The fundamental rationalist critique of the Empirical Paradigm is that it is patently obvious that employing a more systematic, methodical, logical process should improve outcomes [7,23,119,172,173]. The fundamental empiricist critique of the Rational Paradigm is that there is no convincing evidence that following more systematic, methodical, logical processes is helpful or even possible [3,5,9,12]. As the Rational Paradigm is grounded in Rationalist epistemology, its adherents are skeptical of empirical evidence [23]; similarly, as the Empirical Paradigm is grounded in empiricist epistemology, its adherents are skeptical of appeals to intuition and common sense [5]. In other words, scholars in different paradigms talk past each other and struggle to communicate or find common ground.
3) Many reasonable professionals, who would never buy a homeopathic remedy (because a few testimonials obviously do not constitute sound evidence of effectiveness) will adopt a software method or practice based on nothing other than a few testimonials [174,175]. Both practitioners and researchers should demand direct empirical evaluation of the effectiveness of all proposed methods, tools, models, standards and techniques (cf. [111,176]). When someone argues that basic standards of evidence should not apply to their research, call this what it is: the special pleading fallacy

[177]. Meanwhile, peer reviewers should avoid criticizing or rejecting empirical work for contradicting non-empirical legacy concepts.

4) The Rational Paradigm leads professionals "to demand up-front statements of design requirements" and "to make contracts with one another on [this] basis", increasing risk [5]. The Empirical Paradigm reveals why: as the goals and desiderata coevolve with the emerging software product, many projects drift away from their contracts. This drift creates a paradox for the developers: deliver exactly what the contract says for limited stakeholder benefits (and possible harms), or maximize stakeholder benefits and risk breach-of-contract litigation. Firms should therefore consider alternative arrangements including in-house development or ongoing contracts.
5) The Rational Paradigm contributes to the well-known tension between managers attempting to drive projects through cost estimates and software professionals who cannot accurately estimate costs [88]. Developers underestimate effort by 30–40% on average [178] as they rarely have sufficient information to gauge project difficulty [18]. The Empirical Paradigm reveals that design is an unpredictable, creative process, for which accounting-based control is ineffective.
6) Rational Paradigm assumptions permeate IS2010 [70] and SE2014 [179], the undergraduate model curricula for information systems and software engineering, respectively. Both curricula discuss requirements and lifecycles in depth; neither mention Reflection-in-Action, coevolution, amethodical development or any theories of SE or design (cf. [180]). Nonempirical legacy concepts including the Waterfall Model and Project Triangle should be dropped from curricula to make room for evidenced-based concepts, models and theories, just like in all of the other social and applied sciences.

5.2. Limitations and objections

The above recommendations should be considered in light of several limitations. This is obviously not an empirical paper. We cannot refute Rationalism with an empirical study because of The Superiority of Reason Thesis. Rather, this paper uses philosophical deconstruction to understand a worldview and body of literature. The underlying analysis may be biased by the empiricist leanings of the author. However, its claims regarding the literature are straightforwardly verifiable. For example, anyone can verify the claim that Schön [9] presents empirical evidence to support his view of designers while Simon [7] does not simply by reviewing the books in question.

Furthermore, to avoid overcomplicating the issues and obfuscating the core message, this paper does not attempt to enumerate all elements of the two paradigms, discuss related trends including the ongoing shift from modernism to postmodernism [181], or organize *all* SE literature into Rationalist and Empiricist camps. It also does not attempt a complete review of contemporary epistemology and philosophy of science (particularly Bayesian Epistemology and Critical Realism) not only because of space limitations but also because this, again, would overcomplicate the core argument. Moreover, other paradigms not considered here may exist, or may arise in the future.

This article's strong criticism of the Rational Paradigm may precipitate several objections:

1) Many proponents of the Rational Paradigm are well-known, respected scholars. Surely, they could not be so wrong? Of course, this is the ad hominem fallacy. Herbert Simon, for example, won the Sveriges Riksbank Prize in Economic Sciences, but his ideas about how designers work have not been supported by empirical research. No one is right all of the time.
2) If the Rational Paradigm is so wrong and harmful, why has it persisted for decades? Of course, this precisely commits the appeal to ancient wisdom – a naturalistic fallacy holding that a doctrine must be true if it persisted for a long time, regardless of empirical evidence. Many beliefs, including the idea that Royce's taxonomy describes software development and that different parts of the tongue sense different tastes have persisted for decades despite being false.
3) Moreover, seeing the Rational and Empirical Paradigms stake out seemingly extreme positions, one might expect the truth to lie in between. Of course, this precisely commits the middle ground fallacy. There is no truth in the middle ground between the Empirical Paradigm and the Rational Paradigm any more than between evolution and creationism or between medicine and homeopathy. One represents the best understanding we have of a complex phenomenon, the other is a snake pit of false narratives. This paper does not present a fair and balanced analysis of the pros and cons of each paradigm because any such analysis would be deeply misleading. Rationalism runs contrary to the full weight of modern science and has no place in the SE community. While the SE community has made significant strides toward Empiricism insofar as expecting empirical evaluations of proposed artifacts, much progress is still needed not only in demanding more meaningful evaluations but also in abandoning legacy, non-empirical concepts.

5.3. Future research

These contributions and limitations suggest five avenues for future research:

1. *Observational studies of expert developers*. Many of the empirical studies of expert designers reviewed in this paper studied architects, industrial designers, products designers and (non-software) engineers. More longitudinal, observational studies of software professionals (e.g. [182,183]) are needed to reveal crucial phenomena unique to more malleable artifacts.

2. *Theories of development processes*. Our understanding of software processes is currently rooted in nonempirical methods [104]. We can address this by developing theories of the various processes involved in building software systems [92], including the process of forming SE teams, the process through which agility arises [113], the process of finding a bug and the software development process in general [10].
3. *Theories of development practices*. While some practices (e.g. pair programming) have been extensively validated [117], we have no theory to tell us when to use what practices or how different practices interact. How, for instance, does use case analysis interact with persona analysis?
4. *Evidence-based practice*. Evidence-based practice refers to a form of professional activity in which academics investigate practically relevant problems and professional activity is guided by systematic literature reviews. It represents an intense commitment to empiricism and would alleviate many of the problems discussed above. While evidence-based software engineering has gained notoriety [184], more rigorous, primary research is needed to support more useful reviews.
5. *Consideration of recent philosophical advances*. Contemporary advances in epistemology, ontology and philosophy of science, including Critical Realism [185], Bayesian epistemology [186], and the processes driving scientific consensus [187], may have important ramifications for software engineering. By illuminating the influence of Rationalism on the SE discourse, this paper should facilitate further analysis on these and related topics.

5.4. Summary

In conclusion, SE and design research is divided into two disparate groups with different beliefs, values, techniques and assumptions. One group, the Rational Design Paradigm, assumes that software design should be a logical, methodical, meticulously planned process of searching for a satisfactory solution to a given problem. The other group, The Empirical Design Paradigm, assumes that goals, preferences and solution candidates all emerge together from working with stakeholders and building a product, and that design is predominately improvised and amethodical. While there are many sound arguments supporting the Rational Paradigm, the balance of empirical evidence supports the Empirical Design Paradigm. Despite this evidence, the Rational Design Paradigm continues to dominate software engineering standards, methods, curricula and much of the scholarly discourse. The continued dominance of the Rational Paradigm undermines the SE community's scientific credibility; facilitates the spread, adoption and use of ineffective tools and techniques; and undermines software engineering education.

References

[1] K. Dorst, J. Dijkhuis, Comparing paradigms for describing design activity, Des. Stud. 16 (1995) 261–274.
[2] K. Dorst, Design problems and design paradoxes, Des. Issues 22 (2006) 4–17, https://doi.org/10.1162/desi.2006.22.3.4.
[3] K. Dorst, Describing Design: A Comparison of Paradigms, PhD Dissertation, Delft University of Technology, 1997.
[4] T.S. Kuhn, The Structure of Scientific Revolutions, 2nd ed., University of Chicago Press, 1996.
[5] F.P. Brooks, The Design of Design: Essays from a Computer Scientist, Addison-Wesley Professional, 2010.
[6] G. Pahl, W. Beitz, Engineering Design: A Systematic Approach, Springer-Verlag, London, 1996.
[7] H.A. Simon, The Sciences of the Artificial, 3rd ed., MIT Press, Cambridge, MA, USA, 1996.
[8] W. Royce, Managing the development of large software systems, in: Proceedings of WESCON, IEEE, Los Angeles, USA, 1970, pp. 1–9.
[9] D.A. Schön, The Reflective Practitioner: How Professionals Think in Action, Basic Books, USA, 1983.
[10] P. Ralph, The sensemaking–coevolution–implementation theory of software design, Sci. Comput. Program. 101 (2015) 21–41, https://doi.org/10.1016/j.scico.2014.11.007.
[11] K. Dorst, N. Cross, Creativity in the design process: co-evolution of problem–solution, Des. Stud. 22 (2001) 425–437, https://doi.org/10.1016/S0142-694X(01)00009-6.
[12] N. Cross, K. Dorst, N. Roozenburg, Research in Design Thinking, Delft University Press, 1992.
[13] R. Kling, Social analyses of computing: theoretical perspectives in recent empirical research, ACM Comput. Surv. 12 (1980) 61–110, https://doi.org/10.1145/356802.356806.
[14] N. Levina, Collaborating on multiparty information systems development projects: a collective reflection-in-action view, Inf. Syst. Res. 16 (2005) 109–130, https://doi.org/10.1287/isre.1050.0055.
[15] P. Checkland, Systems Thinking, Systems Practice, Wiley, Chichester, 1999.
[16] V. Rajlich, Changing the paradigm of software engineering, Commun. ACM 49 (2006) 67–70, https://doi.org/10.1145/1145287.1145289.
[17] R. Weber, The rhetoric of positivism versus interpretivism: a personal view, MIS Q. 28 (2004) iii–xii.
[18] P. Ralph, Y. Wand, A proposal for a formal definition of the design concept, in: K. Lyytinen, P. Loucopoulos, J. Mylopoulos, W. Robinson (Eds.), Design Requirements Engineering: a Ten-Year Perspective, Springer-Verlag, Cleveland, OH, USA, 2009, pp. 103–136.
[19] P. Freeman, D. Hart, A science of design for software-intensive systems, Commun. ACM 47 (2004) 19–21.
[20] W. Demopoulos, On applying learnability theory to the rationalism-empiricism controversy, in: Learnability and Linguistic Theory, Springer, Dordrecht, Netherlands, 1989, pp. 77–88.
[21] P. Markie, Rationalism vs. empiricism, in: Stanford Encyclopedia of Philosophy, Stanford University, 2013, https://plato.stanford.edu/entries/rationalism-empiricism/.
[22] R. Hoffmann, Mixed strategies in the mugging game, Ration. Soc. 13 (2001) 205–212, https://doi.org/10.1177/104346301013002003.
[23] D.L. Parnas, The limits of empirical studies of software engineering, in: Proceedings of the 2003 International Symposium on Empirical Software Engineering, 2003, pp. 2–5.
[24] V.J. Cook, M. Newson, Chomsky's Universal Grammar: An Introduction, 3rd ed., Wiley-Blackwell, Oxford, UK, 1996.
[25] N. Chomsky, R.S. Cohen, M.W. Wartofsky, Recent contributions to the theory of innate ideas: summary of oral presentation, Synthese 17 (1967) 2–11.
[26] J. Cottingham, Rationalism, Paladin, London, UK, 1984.
[27] W. James, Essays in Radical Empiricism, Harvard University Press, 1976.
[28] T. McGrew, M. Alspector-Kelly, F. Allhoff, Philosophy of Science: An Historical Anthology, Wiley, 2009.

[29] J. Mingers, Realizing information systems: critical realism as an underpinning philosophy for information systems, Inf. Organ. 14 (2004) 87–103, https://doi.org/10.1016/j.infoandorg.2003.06.001.
[30] A.F. Chalmers, What Is This Thing Called Science, 3rd ed., Hackett Publishing, Indianapolis, 1999.
[31] T. Lähdesmäki, P. Hurme, R. Koskimaa, L. Mikkola, T. Himberg, Philosophy of science, mapping research methods, https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/en/methodmap/philosophy-of-science/philosophy-of-science?set_language=en&cl=en, 2010. (Accessed 5 January 2015).
[32] A. Newell, H. Simon, Human Problem Solving, Prentice-Hall, Inc., 1972.
[33] N. Cross, Design cognition: results from protocol and other empirical studies of design activity, in: Design Knowing and Learning: Cognition in Design Education, 2001, pp. 9–103, chap. 7.
[34] C. Zannier, M. Chiasson, F. Maurer, A model of design decision making based on empirical results of interviews with software designers, Inf. Softw. Technol. 49 (2007) 637–653, https://doi.org/10.1016/j.infsof.2007.02.010.
[35] P. Ralph, E. Tempero, Characteristics of decision-making during coding, in: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, ACM, Limerick, Ireland, 2016.
[36] T. Mens, T. Tourwé, A survey of software refactoring, IEEE Trans. Softw. Eng. 30 (2004) 126–139, https://doi.org/10.1109/TSE.2004.1265817.
[37] C. Hookway, Pragmatism, in: Stanford Encyclopedia of Philosophy, Stanford Encyclopedia of Philosophy, 2013.
[38] N. Cross, Descriptive models of creative design: application to an example, Des. Stud. 18 (1997) 427–440.
[39] N. Cross, Design cognition: results from protocol and other empirical studies of design activity, in: C. Eastman, W.C. Newstetter, M. McCracken (Eds.), Design Knowing and Learning: Cognition in Design Education, Elsevier Science, Oxford, UK, 2001, pp. 79–103.
[40] H.A. Simon, From substantive to procedural rationality, in: 25 Years of Economic Theory, Springer US, Boston, MA, 1976, pp. 65–86.
[41] L. Suchman, Human–Machine Reconfigurations: Plans and Situated Actions, Cambridge University Press, 2007.
[42] L. Suchman, Plans and Situated Actions: The Problem of Human–Machine Communication, Cambridge University Press, 1987.
[43] G. Polya, How to Solve It: A New Aspect of Mathematical Method, 2nd ed., Princeton University Press, Princeton, New Jersey, USA, 1957.
[44] Y. Zheng, W. Venters, T. Cornford, Collective agility, paradox and organizational improvisation: the development of a particle physics grid, Inf. Syst. J. 21 (2011) 303–333, https://doi.org/10.1111/j.1365-2575.2010.00360.x.
[45] W. Visser, More or less following a plan during design: opportunistic deviations in specification, Int. J. Man-Mach. Stud. 33 (1990) 247–278, https://doi.org/10.1016/S0020-7373(05)80119-1.
[46] K. Weick, Sensemaking in Organizations, Sage, Thousand Oaks, CA, USA, 1995.
[47] I. Stigliani, D. Ravasi, Organizing thoughts and connecting brains: material practices and the transition from individual to group-level prospective sensemaking, Acad. Manag. J. 55 (2012) 1232–1259, https://doi.org/10.5465/amj.2010.0890.
[48] A. Wright, The role of scenarios as prospective sensemaking devices, Manag. Decis. 43 (2012) 86–101, https://doi.org/10.1108/00251740510572506.
[49] L.C. Rodriguez, M. Mora, M.V. Martin, R. O'Connor, F. Alvarez, Process models of SDLCs: comparison and evolution, in: M.R. Syed, S.N. Syed (Eds.), Handbook of Research on Modern Systems Analysis and Design Technologies and Applications, 2009, pp. 76–89.
[50] A. Tiwana, M. Keil, Control in internal and outsourced software projects, J. Manag. Inf. Syst. 26 (2009) 9–44.
[51] L.J. Kirsch, V. Sambamurthy, D.-G. Ko, R.L. Purvis, Controlling information systems development projects: the view from the client, Manag. Sci. 48 (2002) 484–498.
[52] P. Stacey, J. Nandhakumar, A temporal perspective of the computer game development process, Inf. Syst. J. 19 (2009) 479–497, https://doi.org/10.1111/j.1365-2575.2007.00273.x.
[53] M.W. Lewis, Exploring paradox: toward a more comprehensive guide, Acad. Manag. Rev. 25 (2000) 760–776.
[54] C.F. Kurtz, D.J. Snowden, The new dynamics of strategy: sense-making in a complex and complicated world, IBM Syst. J. 42 (2003) 462–483.
[55] J.S. Gero, Design prototypes: a knowledge representation schema for design, AI Mag. 11 (1990) 26–36.
[56] P. Kruchten, Casting software design in the function–behavior–structure framework, IEEE Softw. 22 (2005) 52–58.
[57] J.S. Gero, U. Kannengiesser, An ontological model of emergent design in software engineering, in: Proceedings of the 16th International Conference on Engineering Design, Paris, France, 2007.
[58] P. Ralph, Fundamentals of Software Design Science, PhD Dissertation, University of British Columbia, 2010.
[59] R. Atkinson, Project management: cost, time and quality, two best guesses and a phenomenon, it's time to accept other success criteria, Int. J. Proj. Manag. 17 (1999) 337–342.
[60] W.H. DeLone, E.R. McLean, The DeLone and McLean model of information systems success: a ten-year update, J. Manag. Inf. Syst. 19 (2003) 9–30.
[61] P. Ralph, P. Kelly, The dimensions of software engineering success, in: Proceedings of the International Conference on Software Engineering, ACM, Hyderabad, India, 2014, pp. 24–35.
[62] B. Flyvbjerg, Design by deception: the politics of major project approval, Harv. Des. Mag. 22 (2005) 50–59.
[63] A. Herrmann, B. Paech, Practical challenges of requirements prioritization based on risk estimation, Empir. Softw. Eng. 14 (2009) 644–684, https://doi.org/10.1007/s10664-009-9105-0.
[64] J. Marchant, C. Tjortjis, M. Turega, A metric of confidence in requirements gathered from legacy systems: two industrial case studies, in: Proceedings of the 10th Conference on Software Maintenance and Reengineering, IEEE, 2006.
[65] J.A. Goguen, C. Linde, Techniques for requirements elicitation, in: Proceedings of the IEEE International Symposium on Requirements Engineering, 1993, pp. 152–164.
[66] IEEE, ISO/IEC/IEEE Standard 29148-2011: Systems and software engineering – Life cycle processes – Requirements engineering, 2011, https://doi.org/10.1109/IEEESTD.2011.6146379.
[67] I. Jacobson, G. Booch, J. Rumbaugh, The Unified Software Development Process, Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1999.
[68] P. Kruchten, The Rational Unified Process: An Introduction, 3rd ed., Addison-Wesley Professional, 2004.
[69] M. Ardis, D. Budgen, G. Hislop, J. Offutt, M. Sebern, W. Visser, Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE/ACM, 2015.
[70] H. Topi, J.S. Valacich, R.T. Wright, K.M. Kaiser, J.F. Nunamaker, J.C. Sipior, et al., IS 2010: curriculum guidelines for undergraduate degree programs in information systems, Commun. Assoc. Inf. Syst. 26 (2010) 18.
[71] P. Berger, T. Luckmann, The Social Construction of Reality: A Treatise in the Sociology of Knowledge, Penguin, London, 1966.
[72] S. Lichtenstein, P. Slovic, The Construction of Preference, Cambridge University Press, 2006.
[73] N. Maiden, S. Jones, K. Karlsen, R. Neill, K. Zachos, A. Milne, Requirements engineering as creative problem solving: a research agenda for idea finding, in: Proceedings of the 18th IEEE International Requirements Engineering Conference, IEEE, 2010, pp. 57–66.
[74] S. Chakraborty, S. Sarker, S. Sarker, An exploration into the process of requirements elicitation: a grounded approach, J. Assoc. Inf. Syst. 11 (2010) 1.
[75] P. Ralph, The illusion of requirements in software development, Requir. Eng. 18 (2013) 293–296, https://doi.org/10.1007/s00766-012-0161-4.
[76] R. Mohanani, P. Ralph, B. Shreeve, Requirements fixation, in: Proceedings of the International Conference on Software Engineering, IEEE, Hyderabad, India, 2014, pp. 895–906.
[77] P. Ralph, R. Mohanani, Is requirements engineering inherently counterproductive?, in: Proceedings of the 5th International Workshop on the Twin Peaks of Requirements and Architecture, Florence, Italy, 2015, pp. 20–23.
[78] N. Cross, Design Thinking: Understanding How Designers Think and Work, Berg, Oxford, UK, 2011.

[79] S.K. Sim, A.H.B. Duffy, Towards an ontology of generic engineering design activities, Res. Eng. Des. 14 (2003) 200–223.
[80] B. Curtis, M.I. Kellner, J. Over, Process modeling, Comm. ACM 35 (1992) 75–90.
[81] A.H. Van de Ven, Engaged Scholarship: A Guide for Organizational and Social Research, Oxford University Press, Oxford, UK, 2007.
[82] C.W. Alexander, Notes on the Synthesis of Form, Harvard University Press, 1964.
[83] R. Sabherwal, M. Newman, Persistence and change in system development: a dialectical view, J. Inf. Technol. 18 (2003) 69–92, https://doi.org/10.1080/0268396032000101144.
[84] P. Stacey, J. Nandhakumar, Opening up to agile games development, Commun. ACM 51 (2008) 143–146, https://doi.org/10.1145/1409360.1409387.
[85] E. Bjarnason, K. Smolander, E. Engström, P. Runeson, Alignment practices affect distances in software development: a theory and a model, in: Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering, ACM, 2014, pp. 21–31.
[86] B. Boehm, A spiral model of software development and enhancement, Computer 21 (1988) 61–72.
[87] K. Forsberg, H. Mooz, The relationship of system engineering to the project cycle, Engin. Manag. J. 4 (1992) 36–43.
[88] K. Beck, Extreme Programming eXplained: Embrace Change, 2nd ed., Addison Wesley, Boston, MA, USA, 2005.
[89] OMG, Essence – Kernel and Language for Software Engineering Methods, 1st ed., Object Management Group, 2014, http://www.omg.org/cgi-bin/doc?ad/2012-11-01.
[90] P.E. Vermaas, K. Dorst, On the conceptual framework of John Gero's FBS-model and the prescriptive aims of design methodology, Des. Stud. 28 (2007) 133–157.
[91] A.H. Van de Ven, M.S. Poole, Explaining development and change in organizations, Acad. Manag. Rev. 20 (1995) 510–540.
[92] P. Ralph, Developing and evaluating software engineering process theories, in: Proceedings of the International Conference on Software Engineering, IEEE Press, Florence, Italy, 2015, pp. 20–31.
[93] B. Fitzgerald, The transformation of open source software, MIS Q. 30 (2006) 587–598.
[94] K. Ewusi-Mensah, Software Development Failures, MIT Press, Cambridge, MA, USA, 2003.
[95] K. Laudon, J. Laudon, M. Brabston, Management Information Systems: Managing the Digital Firm, fourth Canadian ed., Pearson, Prentice Hall, Toronto, 2009.
[96] P. Bourque, R.E. Fairley (Eds.), Guide to the Software Engineering Body of Knowledge, Version 3.0, IEEE Computer Society, 2014, www.swebok.org.
[97] IEEE, IEEE Standard 830-1998: Recommended Practice for Software Requirements Specifications, 1998.
[98] J. Parsons, Y. Wand, Using cognitive principles to guide classification in information systems modeling, MIS Q. 32 (2008) 839–868.
[99] W.P. Stevens, G.J. Myers, L.L. Constantine, Structured design, IBM Syst. J. 13 (1974) 115–139.
[100] T. Fullerton, C. Swain, S. Hoffman, Game Design Workshop: Designing, Prototyping, & Playtesting Games, Taylor & Francis, 2004.
[101] K. Beck, Test Driven Development: By Example, Addison-Wesley Professional, Boston, MA, USA, 2002.
[102] P. Ralph, Software engineering process theory: a multi-method comparison of sensemaking–coevolution–implementation theory and function–behavior–structure theory, Inf. Softw. Technol. 70 (2016) 232–250, https://doi.org/10.1016/j.infsof.2015.06.010.
[103] J. Grudin, J. Pruitt, Personas, participatory design and product development: an infrastructure for engagement, in: Proceedings of the Participatory Design Conference, 2002, pp. 141–161.
[104] D.P. Truex, R. Baskerville, J. Travis, Amethodical systems development: the deferred meaning of systems development methods, Account. Manag. Inf. Technol. 10 (2000) 53–79.
[105] K. Schwaber, M. Beedle, Agile Software Development with Scrum, Prentice Hall, 2001.
[106] M. Poppendieck, T. Poppendieck, Lean Software Development: An Agile Toolkit, Addison-Wesley Professional, 2003.
[107] P. Kruchten, The Rational Unified Process: An Introduction, Addison-Wesley Professional, 2004.
[108] Great Britain Office of Government Commerce, Managing Successful Projects with PRINCE2, Stationery Office Books, 2009.
[109] A. Hevner, S. Chatterjee, Design Research in Information Systems: Theory and Practice, Springer, 2010.
[110] A. Hevner, S.T. March, J. Park, S. Ram, Design science in information systems research, MIS Q. 28 (2004) 75–105.
[111] S.T. March, G.F. Smith, Design and natural science research on information technology, Decis. Support Syst. 15 (1995) 251–266.
[112] J. Wynekoop, N. Russo, Studying system development methodologies: an examination of research methods, Inf. Syst. J. 7 (1997) 47–65.
[113] M. Wufka, P. Ralph, Explaining agility with a process theory of change, in: Proceedings of the Agile Conference, Washington, DC, USA, 2015, pp. 60–64.
[114] K. Conboy, Agility from first principles: reconstructing the concept of agility in information systems development, Inf. Syst. Res. 20 (2009) 329–354.
[115] K. Schwaber, J. Sutherland, The scrum guide, http://www.scrum.org/scrumguides/, 2010 (Accessed 2 July 2012).
[116] K. Beck, M. Beedle, A. van Bennekum, A. Cockburn, W. Cunningham, M. Fowler, et al., Manifesto for agile software development, http://www.agilemanifesto.org/, 2001 (Accessed 2 July 2012).
[117] J.E. Hannay, T. Dyba, E. Arisholm, D. Sjøberg, The effectiveness of pair programming: a meta-analysis, Inf. Softw. Technol. 51 (2009) 1110–1122, https://doi.org/10.1016/j.infsof.2009.02.001.
[118] H.A. Simon, Rational choice and the structure of the environment, Psychol. Rev. 63 (1956) 129.
[119] D.L. Parnas, P.C. Clements, A rational design process: how and why to fake it, IEEE Trans. Softw. Eng. 12 (1986) 251–257.
[120] D.L. Parnas, J. Madey, Functional documents for computer systems, Sci. Comput. Program. 25 (1995) 41–61, https://doi.org/10.1016/0167-6423(95)96871-j.
[121] I. Jacobson, G. Booch, J.E. Rumbaugh, The Unified Software Development Process, Addison-Wesley, 1999.
[122] J. Herbsleb, D. Goldenson, A systematic survey of CMM experience and results, in: Proceedings of the 18th International Conference on Software Engineering, 1996, pp. 323–330.
[123] C. Avgerou, T. Cornford, A review of the methodologies movement, J. Inf. Technol. 8 (1993) 277–286.
[124] B. Dobing, J. Parsons, How UML is used, Commun. ACM 49 (2006) 109–113.
[125] M. Petre, UML in practice, in: Proceedings of the 2013 International Conference on Software Engineering, IEEE Press, 2013.
[126] E.A. Whitley, Method-Ism in practice: investigating the relationship between method and understanding in web page design, in: Proceedings of the International Conference on Information Systems, AIS, Helsinki, Finland, 1998, pp. 68–75.
[127] L. Mathiassen, S. Purao, Educating reflective systems developers, Inf. Syst. J. 12 (2002) 81–102, https://doi.org/10.1046/j.1365-2575.2002.00122.x.
[128] J. Bansler, K. Bødker, A reappraisal of structured analysis: design in an organizational context, ACM Trans. Inf. Syst. 11 (1993) 165–193.
[129] D. Avison, G. Fitzgerald, Information systems development, in: W. Currie, R. Galliers (Eds.), Rethinking Management Information Systems: An Interdisciplinary Perspective, Oxford University Press, Oxford, UK, 1999, pp. 250–278.
[130] P. Ralph, Comparing two software design process theories, in: R. Winter, J.L. Zhao, S. Aier (Eds.), Proceedings of the International Conference on Design Science Research in Information Systems and Technology, Springer, St. Gallen, Switzerland, 2010, pp. 139–153.
[131] M. Magni, L. Maruping, M. Hoegl, L. Proserpio, Improvisation and performance in software development teams: the role of geographic dispersion, in: Proceedings of the International Conference on Information Systems, AIS, Paris, France, 2008.
[132] P. Kruchten, The Rational Unified Process: An Introduction, 1st ed., Addison-Wesley Professional, 1998.
[133] A.T. Bahill, F.F. Dean, Discovering system requirements, in: A.P. Sage, W.B. Rouse (Eds.), Handbook of Systems Engineering and Management, 2nd ed., John Wiley & Sons, 2009, pp. 205–266.
[134] D.M. Russell, M.J. Stefik, P. Pirolli, S.K. Card, The cost structure of sensemaking, in: Proceedings of the INTERACT and CHI Conference on Human Factors in Computing Systems, ACM, 1993, pp. 269–276.

[135] A.D. Brown, P. Stacey, J. Nandhakumar, Making sense of sensemaking narratives, Hum. Relat. 61 (2008) 1035–1062, https://doi.org/10.1177/0018726708094858.
[136] M.W. Newell, M.N. Grashina, The Project Management Question and Answer Book, AMACOM Div. American Mgmt. Assn., 2004.
[137] P.C. Fishburn, Utility Theory for Decision Making, Wiley, New York, USA, 1970.
[138] M. Paulk, Capability Maturity Model for Software, Wiley Online Library, 1993.
[139] B. Shore, Systematic biases and culture in project failures, Proj. Manag. J. 39 (2008) 5–16, https://doi.org/10.1002/pmj.20082.
[140] W. Samuelson, R.J. Zeckhauser, Status quo bias in decision making, J. Risk Uncertain. 1 (1988) 7–59.
[141] J. Diaz-Herrera, T. Hillburn (Eds.), Software Engineering 2004: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE/ACM, 2004.
[142] D.T. Campbell, J. Stanley, Experimental and Quasi-Experimental Designs for Research, Houghton Mifflin, Boston, 1963.
[143] W. Trochim, Research Methods Knowledge Base, Atomic Dog Publishing, Cincinnati, OH, USA, 2001.
[144] R.K. Yin, Case Study Research: Design and Methods, 4 ed., Sage, California, USA, 2008, http://www.worldcat.org/title/case-study-research-design-and-methods/oclc/226984696.
[145] K.E. Weick, Educational organizations as loosely coupled systems, Adm. Sci. Q. 21 (1976) 1–19, https://doi.org/10.2307/2391875.
[146] N. Cross, Designerly Ways of Knowing, Springer, 2006.
[147] M. Arvola, H. Artman, Enactments in interaction design: how designers make sketches behave, Artifact 1 (2007) 106–119.
[148] Standish Group, Chaos Database: Chaos Surveys Conducted from 1994 to Fall 2004, 2006.
[149] E. Cardozo, J. Neto, A. Barza, A. França, F. da Silva, SCRUM and productivity in software projects: a systematic literature review, in: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering, EASE, Newcastle, UK, 2010.
[150] E. Stolterman, How system designers think about design and methods: some reflections based on an interview study, Scand. J. Inf. Syst. 3 (1991) 137–150.
[151] J. Nandhakumar, D. Avison, The fiction of methodological development: a field study of information systems development, Inf. Technol. People 12 (1999) 176–191.
[152] E. Whitworth, R. Biddle, Motivation and cohesion in agile teams, in: G. Concas, E. Damiani, M. Scotto, G. Succi (Eds.), Agile Processes in Software Engineering and Extreme Programming, Springer, Berlin/Heidelberg, 2007, pp. 62–69.
[153] A.T. Kocatas, C. Erbas, Extending essence kernel to enact practices at the level of software modules, in: Proceedings of the 3rd SEMAT Workshop on General Theories of Software Engineering, ACM, 2014, pp. 32–35, http://dl.acm.org/citation.cfm?id=2593758.
[154] E.H. Ferneley, P. Sobreperez, Resist, comply or workaround? An examination of different facets of user engagement with information systems, Eur. J. Inf. Syst. 15 (2006) 345–356, https://doi.org/10.1057/palgrave.ejis.3000629.
[155] M.J. Baker, Data collection – questionnaire design, Mark. Rev. 3 (2003) 343–370, https://doi.org/10.1362/146934703322383507.
[156] A. Stefik, S. Hanenberg, Methodological irregularities in programming-language research, Computer 50 (2017) 60–63, https://doi.org/10.1109/MC.2017.3001257.
[157] M.V. Zelkowitz, An update to experimental models for validating computer technology, J. Syst. Softw. 82 (2009) 373–376, https://doi.org/10.1016/j.jss.2008.06.040.
[158] P. Runeson, M. Höst, Guidelines for conducting and reporting case study research in software engineering, Empir. Softw. Eng. 14 (2009) 131–164, https://doi.org/10.1007/s10664-008-9102-8.
[159] K.-J. Stol, P. Ralph, B. Fitzgerald, Grounded theory in software engineering research: a critical review and guidelines, in: Proceedings of the International Conference on Software Engineering, IEEE, Austin, TX, USA, 2016, pp. 120–131.
[160] D. Sjøberg, B. Anda, E. Arisholm, T. Dyba, M. Jørgensen, A. Karahasanovic, E.F. Koren, M. Vokac, Conducting realistic experiments in software engineering, in: ISESE'02 Proc. 2002 Int. Symp. on Empirical Software Engineering, IEEE Comput. Soc., Nara, Japan, 2002, pp. 17–26.
[161] A.J. Ko, T.D. LaToza, M.M. Burnett, A practical guide to controlled experiments of software engineering tools with human participants, Empir. Softw. Eng. 20 (2013) 110–141, https://doi.org/10.1007/s10664-013-9279-3.
[162] A. Rainer, T. Hall, N. Baddoo, Persuading developers to "buy into" software process improvement: a local opinion and empirical evidence, in: ISESE-03, IEEE Comput. Soc., 2003, pp. 326–335.
[163] P. Devanbu, T. Zimmermann, C. Bird, Belief & evidence in empirical software engineering, in: Proceedings of the International Conference on Software Engineering, IEEE, Austin, Texas, USA, 2016, pp. 108–119.
[164] H. Sharp, H. Robinson, M. Woodman, Software engineering: community and culture, IEEE Softw. 17 (2000) 40–47, https://doi.org/10.1109/52.819967.
[165] J.M. Campanario, Commentary on influential books and journal articles initially rejected because of negative referees' evaluations, Sci. Commun. 16 (1995) 304–325, https://doi.org/10.1177/1075547095016003004.
[166] M.J. Mahoney, Publication prejudices: an experimental study of confirmatory bias in the peer review system, Cogn. Ther. Res. 1 (1977) 161–175.
[167] M. Bunge, What is pseudoscience?, Skept. Inq. 9 (1984) 36–46.
[168] T. Sedano, P. Ralph, C. Péraire, Software Development Waste, IEEE Press, 2017.
[169] M. Poppendieck, Lean Software Development, IEEE Computer Society, 2007.
[170] P. Ralph, Improving coverage of design in information systems education, in: Proceedings of the 2012 International Conference on Information Systems, AIS, Orlando, FL, USA, 2012.
[171] R. Collins, Why the social sciences won't become high-consensus, rapid discovery science, in: S. Cole (Ed.), What's Wrong with Sociology?, Transaction Publishers, 2001, pp. 61–85.
[172] D.L. Parnas, Document based rational software development, Knowl.-Based Syst. 22 (2009) 132–141, https://doi.org/10.1016/j.knosys.2008.11.001.
[173] J. Herbsleb, D. Zubrow, D. Goldenson, W. Hayes, M. Paulk, Software quality and the capability maturity model, Commun. ACM 40 (1997) 30–40.
[174] I. Jacobson, B. Meyer, Methods Need Theory, 2009, DrDobbs.com.
[175] P. Johnson, M. Ekstedt, I. Jacobson, Where's the theory for software engineering?, IEEE Softw. 29 (2012) 94–96, https://doi.org/10.1109/MS.2012.127.
[176] J. Venable, J. Pries-Heje, R. Baskerville, A comprehensive framework for evaluation in design science research, in: K. Peffers, M. Rothenberger, B. Kuechler (Eds.), Proceedings of Design Science Research in Information Systems and Technology, in: LNCS, vol. 7286, Springer-Verlag, 2012, pp. 423–438.
[177] T. Damer, Attacking Faulty Reasoning: A Practical Guide to Fallacy-Free Arguments, Cengage Learning, 2008.
[178] K. Molokken, M. Jørgensen, A review of software surveys on software effort estimation, in: Proceedings of the International Symposium on Empirical Software Engineering, ACM-IEEE, 2003, pp. 223–230.
[179] Joint Task Force on Computing Curricula, Software Engineering 2014: Curriculum Guidelines for Undergraduate Degree Programs in Software Engineering, IEEE, ACM, 2015.
[180] P. Ralph, Improving coverage of design in information systems education, in: Proceedings of the International Conference on Software Engineering, IEEE, Orlando, Florida, USA, 2012.
[181] H. Robinson, P. Hall, F. Hovenden, J. Rachel, Postmodern software development, Comput. J. 41 (1998) 363–375, https://doi.org/10.1093/comjnl/41.6.363.
[182] T. Sedano, P. Ralph, C. Péraire, Practice and perception of team code ownership, in: Proceedings of the International Conference on Evaluation and Assessment in Software Engineering, Limerick, Ireland, 2016.
[183] T. Sedano, P. Ralph, C. Péraire, Software development waste, in: Proceedings of the International Conference on Software Engineering, IEEE, Buenos Aires, Argentina, 2017, www.researchgate.net/publication/313360479_Software_Development_Waste.

[184] B.A. Kitchenham, T. Dyba, M. Jørgensen, Evidence-based software engineering, in: ICSE-04, IEEE, 2004, pp. 273–281.
[185] M. Archer, C. Decoteau, P. Gorski, D. Little, D. Porpora, T. Rutzou, et al., What is critical realism? Asatheory.org, http://www.asatheory.org/2/post/2016/12/what-is-critical-realism.html, 2016 (Accessed 14 April 2017).
[186] W. Talbott, Bayesian epistemology, in: Stanford Encyclopedia of Philosophy, Stanford, USA, 2008, https://plato.stanford.edu/entries/epistemology-bayesian/.
[187] B. Latour, Science in Action: How to Follow Scientists and Engineers Through Society, Harvard University Press, 1987.