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/© 2018 Elsevier B.V. All rights reserved.

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