Information and Software Technology 87 (2017) 46–61

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof

A design theory for software engineering

Jon G. Hall*, Lucia Rapanotti

School of Computing and Communications, The Open University, Milton Keynes, MK7 6AA, UK

ARTICLE INFO

Article history:
Received 10 June 2016
Revised 17 January 2017
Accepted 24 January 2017
Available online 6 February 2017

Keywords:
Software engineering
Design theory
General engineering
Problem orientation
Problem solving

ABSTRACT

Context: Software Engineering is a discipline that has been shaped by over 50 years of practice. Many have argued that its theoretical basis has been slow to develop and that, in fact, a substantial theory of Software Engineering is still lacking.

Objective: We propose a design theory for Software Engineering as a contribution to the debate. Having done this, we extend it to a design theory for socio-technical systems.

Method: We elaborate our theory based on Gregor's influential 'meta-theoretical' exploration of the structural nature of a theory in the discipline of Information Systems, with particular attention to ontological and epistemological arguments.

Results: We argue how, from an ontological perspective, our theory embodies a view of Software Engineering as the practice of framing, representing and transforming Software Engineering problems. As such, theory statements concern the characterisation of individual problems and how problems relate and transform to other problems as part of an iterative, potentially backtracking, problem solving process, accounting for the way Software Engineering transforms the physical world to meet a recognised need. From an epistemological perspective, we argue how the theory has developed through research cycles including both theory-then-(empirical-)research and (empirical-)research-then-theory strategies spanning over a decade; both theoretical statements and related empirical evidence are included.

Conclusion: The resulting theory provides descriptions and explanations for many phenomena observed in Software Engineering and in the combination of software with other technologies, and embodies analytic, explanatory and predictive properties. There are however acknowledged limitations and current research to overcome them is outlined.

1. Introduction

Software Engineering (SE) is a discipline that has been shaped by over 50 years of practice. Driven primarily by the needs of industry, a theoretical basis has been slow to develop. Recently, Johnson et al. [1] have argued the need for theories which can address 'significant' questions within SE. Their main criticisms are that existing theories tend to be small, addressing limited sets of phenomena, very often implicit and only casually introduced by authors, with little academic discussion or rigorous evaluation within the community. Undoubtably, their arguments have stirred some debate in the wider SE community, and perhaps have been the catalyst for a renewed interest in the theoretical foundation of the discipline.

Our contribution is a design theory for SE which locates software as a solution within problem solving. The theory has developed from our work on Problem Oriented Engineering (POE), a practical engineering framework with an accumulated body of work spanning over a decade, evaluated and validated through a number of real-world engineering case studies. In relation to previous publications, one novel contribution is to make explicit the theory implicit in the definition. In fact, we contribute two design theories. The first concerns problems for which software is exclusively the solution; this is obviously limited given that other engineered artefacts, for instance, end-user documentation or training materials, will usually be needed as part of a solution. The second theory remedies this limitation.

Our presentation is informed by Gregor's [3] 'meta-theoretical' exploration of Information Systems (IS), particularly her ontological, epistemological and domain questions.

The paper is organised as follows. Section 2 recalls the debate on the need for SE theories. Section 3 discusses the theoretical provenance of our theory, followed by its detailed presentation

* Corresponding author.
E-mail addresses: jon.hall@open.ac.uk (J.G. Hall), lucia.rapanotti@open.ac.uk (L. Rapanotti).

http://dx.doi.org/10.1016/j.infsof.2017.01.010
0950-5849/

in Sections 4 and 5. An evaluation and discussion of ongoing research is given in Section 6, while Section 7 discusses related work. Section 8 concludes the paper.

2. Background

2.1. The recent debate for on theory in SE

Johnson et al. [1] constrain neither the form that a significant SE theory should have nor how it should be generated. In contrast, Adolph and Kruchten [4] argue that an SE theory 'must be useful to practitioners and explain the phenomena they are experiencing,' whence proposing an empirical approach. Similarly, Ralph [5] argues for the usefulness of *process theories* in SE, providing both a taxonomy of process theory types and examples of where such theories could be beneficial.

A complementary stance is taken by Staples [6], who proposes that, alongside process theories, what SE needs are *product theories* if engineering concerns, such as performance, are to be taken into account. Accordingly, such product theories may not be exact, but only provide some conservative approximations to support assurance about the use of artefacts, i.e., they need only be general and precise enough to reason about whether an artefact meets acceptable requirements for use. A different perspective yet is taken by Smolander and Paivarinta [7], who argue for the need of *theories of practice* focused on design and development practices, using a reflection-in-action approach to theory generation.

Wieringa [8] argues that a desire for universal theory leads to theoretical idealisations unusable in practice. Instead, he suggests that 'middle-range' theories are more usable by practitioners. The case for middle-range theories is also made by Stol and Fitzgerald [9], 10], who describe them as stepping stones towards a more general and inclusive theory.

More generally, the community is trying to come to terms with what is meant by 'theory.' There is some agreement on what is not considered a theory (e.g., [11]) and much current thinking aligns closely with [3], which constitutes the frame of reference for this paper.

Finally, Ralph [12], Johnson and Ekstedt [13] and others have considered how existing theories, such as complexity and cognition theories, might contribute to a general SE theory.

It appears that current work is speculative: we are still a long way away from an agreement on what the characteristics of a SE theory should be.

2.2. Gregor's meta-theoretical model

Gregor [3] explores IS theory. Given the strong relation between SE and IS, much of her analysis extends naturally to SE and has been adopted in that community.

Gregor's exploration asks four distinct categories of question: core *domain* questions — the phenomena of interest and the boundary of the discipline; *structural* or *ontological* questions — the nature of theory and the form that knowledge contributions make; *epistemological* questions — how the theory is constructed, the research method used and the way knowledge is accumulated; and *socio-political* questions — concerning the socio-political context in which a theory was developed.

Gregor's argues that an IS theory should be able to link natural, social and artificial worlds, drawing upon their respective design sciences, and that a wide view of theory should be taken. This also makes sense for SE and reflects the current SE community view. Gregor also sees no requirement to commit to one specific epistemological view, also mirrored in the current SE debate.

Gregor [3] concludes that some combination of the following is essential:

- description and analysis: to describe phenomena of interest and related constructs, to generalise them within an identified scope, and to analyse their relationships;
- explanation: to comment on 'how, why, and when things happened, relying on varying views of causality and methods for argumentation';
- prediction: to predict what will happen in the future under specified conditions; and
- prescription: to allow the definition of methods or "recipes for doing" which, if followed, will make theory predictions true under specified conditions.

The last quality is notable in relating theory and method: although methods can follow theories, they are not the same thing [5].

3. Research cycle and theory conceptualisation

Our design theory is based on a body of work accumulated in over a decade of research into a design-theoretic approach to problem solving in the context of software and systems engineering. The research cycle adopted has included both theory-then-(empirical-)research and (empirical-)research-then-theory strategies, starting from an initial theoretical proposition [14]. We present our theory in detail alongside a discussion of the empirical evidence accumulated in its support, and of aspects for which both theoretical and empirical work is still required.

3.1. Theoretical provenance

G.F.C Rogers defined engineering as [15]:

> the practice of organising the design and construction of any artifice[1] which transforms the physical world around us to meet some recognised need.

Rogers' definition places engineering in the problem solving domain, the problem being, given a real world environment and need, to design and construct an artifice with the requisite properties.

This definition specialises easily to *Software* Engineering simply by taking the artifice to be software, giving

> software engineering is the practice of organising the design and construction of software which transforms the physical world around us to meet a recognised need.

This 'software-as-artifice' perspective is deficient:

- that software is the sole artifice implies that the environment has a particular form, one in which software causes become physical effects, and *vice-versa*;
- we cannot, *ab initio*, know the precise combination of solution technologies that will ultimately satisfy the need, even less that software will be the sole solution technology.

Whereas the first of these can easily be discharged by careful choice of environment, the second is more difficult and motivates our second theory: it is likely that through solution technology combinations, both formal software and non-formal solution domains will need to be combined. But then, as Turski [16] observed:

> There are two fundamental difficulties involved in dealing with non-formal domains (also known as 'the real world'):
>
> 1. Properties they enjoy are not necessarily expressible in any single linguistic system.

[1] Rogers appears to use *artifice* rather than, for instance, *artefact* to emphasise that a solution may have no physical embodiment, as is the case with software.