Science of Computer Programming 101 (2015) 99–123

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

A framework for exploring unifying theories of empirical software engineering

Dewayne E. Perry

Center for Advanced Research in Software Engineering (ARiSE), The University of Texas at Austin, United States

ARTICLE INFO

Article history:
Received 14 October 2013
Received in revised form 9 August 2014
Accepted 3 September 2014
Available online 26 November 2014

Keywords:
Theories and models of software engineering and software engineering research
Model calculus
Model composition
Framework evaluation

ABSTRACT

One of the reasons for the lack of rigor in software engineering compared to physical and behavioral sciences is that the theories that underpin our work, both as software engineers and as software engineering researchers, have not been given enough attention. To provide a step forward towards greater rigor, a framework has been created with which to explore theories of software engineering and software engineering research. This framework provides a simple theory modeling language and model calculus to explore informally described theories and to generate the results of composing modeled theories.

To illustrate and evaluate this framework, a general theory about software engineering is presented and then two simple theories, D and E, are proposed as the basis for laying out a unified theoretical foundation for software engineering and software engineering research. Software Engineering consists of two logical parts: design, and empirical evaluation (both terms used in their broadest senses). Theory D is the theoretical basis for the design part, and theory E is the theoretical basis for empirical evaluation. These two theories are then composed in various ways to lay out a space (a taxonomy, or ontology if you will) for software engineering and software engineering research. Finally, it is claimed that software engineering and software engineering research (both fully integrated with empirical evaluations) are models (in the logical sense) for these atomic and composed theories. To further evaluate the framework, examples are provided of modeling (implicit) theories found in a number of software engineering (theory) papers.

The results of this research are: 1) a scientific elegance in creating larger more complex theories out of simpler theories, 2) an elegant way of explaining the complexity of software engineering and software engineering research, and 3) a theory modeling language and model calculus for composing the resulting theoretical models.

1. Introduction

The motivation for this research is twofold: 1) to establish a unifying foundation for software engineering, and 2) to establish the same rigorous empirical foundations for software engineering that we find in natural and behavioral sciences. As engineering pursuits go, software engineering is still a very young discipline. Researchers and practitioners have not given sufficient attention to the theories that underpin our discipline, leaving those theories implicit rather than clearly delineated. Further, the empirical support for many of these (implicit) theories and decisions about research and practice are often anecdotal rather than rigorously and empirically evaluated. While our understanding of many critical issues in

E-mail address: perry@mail.utexas.edu.

http://dx.doi.org/10.1016/j.scico.2014.11.011
0167-6423/

empirical studies has improved, there is still a significant amount of work needed to be done to reach the level of understanding to be found in behavioral and physical sciences. There are many useful empirical techniques and structures that can be borrowed and adapted from them but there are still aspects to be more fully developed that are unique to software engineering and software engineering research.

In pursuit of these goals, a framework has been created for modeling informally described theories and a model calculus [15] for composing these modeled theories to create more complex theories from simpler ones. This framework provides a scientific elegance in creating more complex theories out of simpler ones, and an elegant way of explaining the complexity of software engineering and software engineering research. Composition is as fundamental in the theoretical pursuits of software engineering as it is in the practical pursuits of constructing software systems.

In the context of a general theory of software engineering (GTSE), this paper addresses the technical part of software engineering: the design, construction, evaluation, and deployment of software systems. This technical part of software engineering is separated out from such aspects as project management, business economics, and strategic planning. While software engineers play multiple roles in the context of both project management and strategic planning, the concern here is about their role creating and evolving software systems. So with respect to a GTSE, the focus is on the technical side of a GTSE. With respect to how this technical focus fits into a GTSE, see Batory and Perry [17] for a proposed theory about the compositional and structural organization of a GTSE.

The basic theory presented here is that Software Engineering consists of two logical parts: design and empirical evaluation (both terms used in their broadest senses). There are two simple theories proposed, D and E, as the basis for laying out a unified theoretical foundation for software engineering and software engineering research [16]. Theory D is the theoretical basis for the design part, and theory E is the theoretical basis for the empirical evaluation part. Models for D and E are then defined, using the presented theory modeling language, and subsequently, the theories D and E are composed using the model calculus in various ways, and the composed models used to explore various ways of thinking about the underlying foundations for software engineering and software engineering research.

Theories D and E abstract away details that undoubtedly differ significantly in actual research and practice; but, as in practice, abstraction is one of our most important intellectual tools for managing complexity. As will be illustrated in what follows, even with the abstract and simple theories D and E, the composition of theories results in significant increases in complexity, just as it does in the composition of software components into software systems. Brooks in his *No Silver Bullet* paper [5], states: "*I believe the hard part of building software to be the specification, design, and testing* [evaluating] *of this conceptual construct, not the labor of representing it and testing the fidelity of the representation....* If this is true, building software will always be hard. There is inherently no silver bullet." Further, he states that "Software entities are more complex for their size than perhaps any other human construct...." A complete general theory of software engineering is at least as equally hard, and at least as equally complex. The envisioned use of the framework and the model calculus is to delineate and explicate the elements of design theories via models, and to clarify the implications of those design models with respect to their evaluations via the resulting taxonomies. The resulting explicitness of both will help us to understand more fully, and reason more carefully, about what is often hidden in their current implicitness.

The paper is organized as follows. The rest of the introduction 1) defines how the terms *theory* and *model* are used in this paper, 2) provides a general overview of theories, 3) summarizes important aspects of science in general, as well as 4) delineating specific differences among natural, behavioral, and artificial sciences. Section 2 provides a formal definition of models and presents the model calculus. Section 3 illustrates this paper's approach with simple theories D and E and their models. Section 4 illustrates the issues in theory and model composition by composing the simple theories D and E in various ways to cover software engineering and software engineering research. Section 5 presents more complex versions of D and E, D′ and E′, introducing the notion of refining a model and exploring the implications for compositions of these more complex theories in the composition E′ : E′ – the evaluations of evaluations. A variety of examples are used to illustrate various aspects in Sections 4 and 5. Section 6 evaluates the utility of the presented approach by using the framework to explore three rather diverse theories and models. Finally, Section 7 summarizes the results of this research.

1.1. Theories and models

The terms "theory" and "model" are used and misused in a variety of ways, often informally and interchangeably. They are used here in a very specific way: a theory (a more or less abstract entity) is reified, represented, satisfied, etc., by a model (a concrete entity).

This view of theories and models is derived in part from Turski and Maibaum [20] where they state *"A specification is rather like a natural science theory of the application domain, but seen as a theory of the corresponding program it enjoys an unmatched status: it is truly a postulative theory, the program is nothing more than an exact embodiment of the specification"*. In this paper, theory is considered to be broader than a specification and, more than likely, less formal.

Models are often used as a representation of a theory. In natural sciences, the model is often a set of mathematical formulas. In logic, a model is an interpretation of a theory and has certain logical properties. Here the purpose is to broaden the notion of a model to be a representation (indeed, a reification) of the theory. The model is of paramount importance in design disciplines as it is the visible manifestation of a theory. Of fundamental importance is the fact that a theory can have an arbitrary number of models.