Science of Computer Programming 101 (2015) 42–65

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

System components of a general theory of software engineering

Anca-Juliana Stoica [a], Kristiaan Pelckmans [a,*], William Rowe [b]

[a] Department of Information Technology, Uppsala University, Uppsala, Sweden
[b] Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL, USA

ARTICLE INFO

Article history:
Received 13 October 2013
Received in revised form 31 August 2014
Accepted 8 September 2014
Available online 25 November 2014

Keywords:
General theory of software engineering
Model systems
Theoretic Decision Framework
Optimal decision-under-risk process
Adaptive software development

ABSTRACT

The contribution of this paper to a general theory of software engineering is twofold: it presents the model system concept, and it integrates the software engineering design process into a decision making theory and a value-based decision-under-risk process. The model system concept is defined as a collection of interconnected and consistent components that work together for defining, developing, and delivering a software system. This model system concept is used to represent the multiple facets of a software engineering project such as stakeholders and models related to domain/environment, success, decision, product, process, and property. The model system concept is derived from software development practices in the industry and academia.
The theoretical decision framework acts as a central governance component for a given software engineering project. Applying this decision framework allows for effectively managing risks and uncertainties related to success in the project building stage. Especially, this puts the design process in an economic perspective, where concepts such as value-of-waiting, value-of-information and possible outcomes can be coped with explicitly. In practice, the decision framework allows for the optimal control of modern adaptive software development. In particular, one can use dynamic programming to find the optimal sequence of decisions to be made considering a defined time horizon. In this way we can relate our contribution to a theory of software engineering to the well-studied areas of automatic control, optimization, decision theory and Bayesian analysis.
Computational case studies exemplify the conceptual innovations proposed in this paper.

1. Introduction

In the philosophy of science, a *theory* is a belief that there is a pattern in phenomena (Craver [39]). In other words, theory means reflexion and identification of an underlying pattern. Moreover, according to the Stanford Encyclopedia of Philosophy [29] *models* are vehicles to explore, to understand, and to learn about the world, where this cognitive function is the basis of the so-called "model-based reasoning". The aim of the present work is to find a way to deploy these core concepts in the science of software engineering.

In a workshop on software engineering research strategies [7], it was emphasized that an important focus of software engineering science should be placed on finding explanatory/predictive models related to the process, product, as well

* Corresponding author.
E-mail addresses: anca.stoica@it.uu.se (A.-J. Stoica), kp@it.uu.se (K. Pelckmans), wrowe001@ufl.edu (W. Rowe).

http://dx.doi.org/10.1016/j.scico.2014.11.008
0167-6423/© 2014 Elsevier B.V. All rights reserved.

as user, and on interactions between models. Further focus should be on applying the corresponding empirical validation technologies and analytic methods to support these models. In our opinion, as well as that of other researchers [7], software engineering as a scientific discipline requires further development in directions such as: use of integrative technologies and methodologies; ability to forecast the effect of software production in a dynamical setting; efficient and reliable (optimal) decision making support and process to face changes in the environment. Specifically, we address the questions of *how to deal with the many involved components*, and *how to dynamically manage the inherent uncertainty (or risk) associated with decisions* in software engineering.

This paper proposes a system-approach to organize the different model-based components into a general theory of software engineering. We call this organization *a model system*. The model system concept is defined as a collection of interconnected and consistent components that work together for defining, developing, and delivering a software system. Components are used to visualize and reason about the essential project-related aspects. These aspects are: (i) the software product, (ii) its associated process, (iii) the non-functional product characteristics, (iv) the product development environment, (v) the major decisions related to these, (vi) the project stakeholders, and (vii) the success criteria. The *model system* concept is derived from software development practices in the industry and academia. Examples of model systems as practical frameworks used in traditional software development are described in Boehm and Port [1], Fisher [2], Gargaro and Peterson [3], Honeywell's Model-Based Software Development [4], Jacobson, Booch and Rumbaugh [5]. More examples of the practical use of models for software development can be found in the context of agile modeling (see for example Ambler [27]), hybrid software development (Stoica [40]) and model-driven development (Swithinbank et al. [28]).

In this paper, our focus is on the following topics. *First*, we show how the different facets of the software development are integrated into a Model System (MS) concept. Our theory supports modern software system engineering as it relates the technical aspects of the system to the stakeholders bringing a considerable amount of points of view, skills, responsibilities, and interests to the interactions. Both engineering and economic endeavors are considered since most software activities are conditional on human creativity and economic constraints. *Second*, we present a theoretical foundation of a reasoning and Decision Framework (DF). Here, optimal decision-making in software development is based on formal dynamic models related to cost, schedule, process, and risk. *Finally*, the decision-under-risk process is applied to stages of the software development process where major decisions have to be taken and major risks have to be managed. The DF allows for reasoning about the dynamic aspects related to decision making under risk. The decisions are typically related to success in adaptive software development such as value judgments, optimal strategies, technical alternatives, innovations, market conditions, and users' demands.

1.1. *Proposed requirements on a general theory of software engineering*

A general theory regarding any topic should contain a number of elements, see for example Levin [24] and Popper [25]. Their applicability to the software engineering discipline is discussed in Johnson, Ekstedt and Jacobson [8], Johnson and Exman [10], and in Johnson and Ekstedt [26]. This subsection ascertains that in the proposed components of a general theory on software engineering, the following elements are properly addressed.

Relation to real world: The Model System (MS) concept includes the model-related components of a software engineering development project. The Decision Framework (DF) helps predicting the influence of the software decision makers' actions on the software development goals.

Support good decision-making: The Decision Framework (DF) provides a theoretical technique to make optimal, strategic project planning decisions based on value of the resulting project. The theory implies a probabilistic decision method in the software process.

Formalisms: The formalization occurs in several ways: i) the external interactions of the DF with the MS components that provide inputs; ii) the internal components of the DF itself, that is the sets of states, alternatives, decisions, consequences. Moreover, the process itself is formally defined as a sequence of decisions and events that evolve over time; iii) these different components are linked together in a dynamic programming approach; iv) the Dynamic Decision Tree gives a visual representation of the above components.

Consistency: The model system itself emphasizes the consistency of its components. In particular the DF is consistent with itself and with the connected model system components.

Measurability: The conceptual framework is translated by formalizing events, decisions and consequences using utilities, probabilities, information, benefits, costs and values.

Predictability: The result of this theory is a technique for optimal strategic decision making. Hereto, we call upon methods of dynamic programming.

1.2. *Related work*

A number of proposals for a general theory on software engineering are described.

Johnson, Ekstedt and Jacobson [8] discuss candidates or collections of propositions that might be suggested for the software engineering core theory, as well as a good definition of a theory, characteristics, and goals. The traditional four criteria of theoretical quality are consistency, correctness, comprehensiveness and precision. The theory guides from random action to intentional design. Interesting avenues are generated by the SEMAT initiative (www.semat.org) and GUTSE (Grand Unified Theory of Software Engineering) [23].