Science of Computer Programming 101 (2015) 66–78

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

Integrating software engineering theory and practice using essence: A case study

Pan-Wei Ng

ARTICLE INFO

Article history:
Received 13 October 2013
Received in revised form 23 February 2014
Accepted 25 June 2014
Available online 25 November 2014

Keywords:
Software engineering
Theory
Kernel
SEMAT
Essence

ABSTRACT

Software engineering is complex and success depends on many inter-related factors. Theory Based Software Engineering (TBSE) is about providing a practical way for software teams to understand the relationships and the influence of these factors to thereby adapt the way they work. This paper proposes an approach to TBSE based on Essence, a software engineering kernel distilled by the SEMAT (Software Engineering Method and Theory) initiative. Essence supports TBSE by providing a domain model that is useful for organizing and relating software engineering factors. Essence also helps make recommended practices precise and actionable to software teams. We provide a step-by-step application of our approach on an industrial software process improvement case study. The case study achieved 21% productivity gains and 58% decrease in defects. But more importantly than these results, it demonstrates the value of Essence in supporting TBSE.

1. Introduction

It is a cliché to say that the world runs on software today. Software permeates almost every aspect of our daily lives. Although software successes are plentiful, failures are not uncommon and there have been numerous reports on software failures. Even IEEE Spectrum prints a long running weekly column entitled “IT Hiccups of the week” run by Charette [1]. But the real tragedy is that most problems are unnecessary and can be avoided [2].

Our position is that if only software teams were to give a little more thought and apply a little more theory, results would improve dramatically. Unfortunately, the situation today is just as Brooks [3] stated many years ago: “In no other discipline is the gulf between best practice and typical practice so wide.” Lethbridge et al. [4] noted that while most practitioners can quickly get up to speed when it comes to software technologies (such as web, mobile, cloud technologies, etc.), it is much harder to raise capabilities in software engineering, such as requirements, analysis, architecture, testing, and project management. Practitioners frequently rely solely on their, often limited, experience and proceed by trial-and-error without any real learning.

1.1. Making theoretical and research results practical

Our goal is to help practitioners make sound and effective decisions that stand firm on a solid theoretical and empirical basis. Evidence Based Software Engineering (EBSE) [5] had been proposed to help practitioners achieve this goal. The idea is simple: A practitioner with a problem formulates the problem as a question to search available evidence from research databases. The evidence collected provides inputs for practitioners to get good decisions and solutions. However, doing EBSE is not simple. Most primary research efforts report their findings very differently. Important context information,

E-mail address: panwei@ivarjacobson.com.

http://dx.doi.org/10.1016/j.scico.2014.11.009
0167-6423/

conclusions and assumptions might be missing or get buried within research reports [6]. It is also common to see results from different research efforts contradict one another. This poses serious challenges for practitioners who are operating under both business and schedule pressures and have little spare time to sift through these research results.

To alleviate this problem, there had been secondary research such as systematic literature reviews to summarize primary research results so that practitioners can get answers quickly to questions on topics such as agile software development [7], test-driven development [8], design patterns [9] and so on. Even reviews about reviews (meta-reviews) [10] exist. However, since no generally accepted framework to organize research results exists, practitioners are left to sift through available data to filter out relevant and applicable information. Glass [11] had lamented that there had been almost no effort to define the realm of applicability of methodologies. The situation is still true even today. Without a comprehensive and widely acceptable theoretical framework it would still be difficult to search and apply relevant research findings.

1.2. The SEMAT initiative

The recent SEMAT (Software Engineering Method and Theory) initiative was founded to bring together industry, research and education to deal with the problem of immature practice in software engineering [12,13]. SEMAT strives towards this goal in two steps, namely:

1. To establish a software engineering language and kernel, and,
2. To establish a general and widely accepted theoretical framework.

The first step had already yielded a language and kernel of software engineering known as Essence. The Essence specification is fast becoming a standard adopted by OMG (Object Management Group) at the time of this writing [14]. Essence captures important success factors of software engineering and makes them practical and actionable to software teams through a multi-dimensional state-based method-independent model of software engineering endeavors. Essence further provides extension mechanisms to deal with project specific factors and practices. Using Essence, a team can describe, analyze and compare different approaches to developing software such as agile versus traditional. Essence has also been demonstrated to provide guidance to both small and large development [15,16].

The second step is about establishing a sound and widely accepted theoretical framework and make it practical to software teams [13,17]. To-date, two conference workshops have been held to get proposals and insights for such a general theoretical framework [18,19]. This paper is part this effort. We believe that practice and theory are tightly knitted and cannot be divorced from one another. Theory requires empirical validation, and practices when repeated many times lays the ground for theory. As it has been said, “there is nothing more practical than good theory [21].” Thus, this paper investigates the theoretical foundation and application of Essence.

1.3. Objectives and overview of paper

In our earlier work [20], we had investigated and demonstrated the value of Essence to organize research findings. In this paper, we go a step further to demonstrate the value of Essence to relate factors affecting software engineering success. We apply an approach, which we call “Theory Based Software Engineering” (TBSE). We organize this paper as follows.

Section 2 gives a brief description of Essence and introduces our approach towards Theory Based Software Engineering (TBSE). We explain the theoretical basis behind Essence and how it provides a novel way to describe software engineering context and organize success factors.

Section 3 demonstrates the use of TBSE in a software process improvement case study of a software product line. This software product line achieved 21% productivity gains and 58% decrease in defects. We provide describe step-by-step how a specific theory is constructed to relate to factors with process improvement objectives. This specific theory is used to devise recommended process improvement actions. Results observed are used to validate the specific theory and thereby prove that the improvements are indeed the result of the recommended actions.

Section 4 discusses the results of this paper including the generality and limitations of our TBSE approach. Finally in Section 5, we discuss contribution to the SEMAT initiative and to a general theory of software engineering.

2. Essence and theory based software engineering (TBSE)

In this section, we briefly present the theoretical foundation underlying Essence and our approach towards Theory Based Software Engineering (TBSE). Everyone has a different idea of what a theory is. In this paper, we define theory as a statement, or set of statements that relates variables of interests and in particular between software development factors and measures of success.

Most software engineers have learnt common theories and laws of physics in high school. For example, the acceleration of a parachute or a falling object is positively influenced by the gravitational pull and negatively by air drag, which is in turned positively influenced by the velocity of the object (see Fig. 1). In Fig. 1, a directed line denotes an effect from one end to another end, with a circle having a sign to show a positive (contribution) or negative (inhibition) influence.