The Journal of Systems and Software 102 (2015) 88–108

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Progressive Outcomes: A framework for maturing in agile software development

Rafaela Mantovani Fontana [a,b,*], Victor Meyer Jr. [a], Sheila Reinehr [a], Andreia Malucelli [a]

[a] Pontifical Catholic University of Paraná (PUCPR), R. Imaculada Conceição, 1155, Prado Velho, 80215-901 Curitiba, PR, Brazil
[b] Federal University of Paraná (UFPR), R. Dr. Alcides Vieira Arcoverde, 1225, Jd. das Américas, 81520-260 Curitiba, PR, Brazil

ARTICLE INFO

Article history:
Received 25 August 2014
Revised 2 December 2014
Accepted 15 December 2014
Available online 22 December 2014

Keywords:
Maturity
Agile software development
Software process improvement
Complex adaptive systems
Ambidexterity

ABSTRACT

Maturity models are used to guide improvements in the software engineering field and a number of maturity models for agile methods have been proposed in the last years. These models differ in their underlying structure prescribing different possible paths to maturity in agile software development, neglecting the fact that agile teams struggle to follow prescribed processes and practices. Our objective, therefore, was to empirically investigate how agile teams evolve to maturity, as a means to conceive a theory for agile software development evolvement that considers agile teams nature. The complex adaptive systems theory was used as a lens for analysis and four case studies were conducted to collect qualitative and quantitative data. As a result, we propose the Progressive Outcomes framework to describe the agile software development maturing process. It is a framework in which people have the central role, ambidexterity is a key ability to maturity, and improvement is guided by outcomes agile teams pursue, instead of prescribed practices.

1. Introduction

Maturity models are tools to describe how an element evolves. This element, which may be a person, an object, or a social system, may use the description provided by maturity models to assess its own situation and find guidance to improve in a specific focus area (Kohlegger et al., 2009). In the software engineering field, process improvement is based mainly on the guidelines given by the Capability Maturity Model Integration for Development –CMMI-DEV (CMMI Product Team, 2010) and the international standard ISO/IEC 15504 (ISO/IEC, 2004). Both CMMI-DEV and ISO/IEC 15504 share the underlying assumption that organizational capabilities must be codified in processes that are previously planned and designed (Maier et al., 2012). The improvement for the organization comes from the definition, institutionalization and quantitative management of these processes (CMMI Product Team, 2010).

A number of agile software development teams have been implementing these process improvement initiatives (Al-Tarawneh et al., 2011; Anderson, 2005; Baker, 2006; Caffery et al., 2008; Cohan and Glazer, 2009; Jakobsen and Johnson, 2008; Lina and Dan, 2012; Lukasiewicz and Miler, 2012; Spoelstra et al., 2011; Sutherland et al., 2007; Tuan and Thang, 2013). The benefits of such initiatives have been recognized as a "magic potion", as they provide a powerful combination of adaptability and predictability (Sutherland et al., 2007). However, if teams are meant to keep agile in the highest maturity levels, the improvement path cannot be based on current established maturity models. The increasing processes definition hinders sustaining agility (Lukasiewicz and Miler, 2012; Paulk, 2001).

If agile methods place people and interaction over processes and tools (Beck et al., 2001; Conboy et al., 2011), the improvement road map for these methods should not be based on processes definition (Fontana et al., 2014). There are, for this reason, a number of agile maturity models proposed in the literature (Leppänen, 2013; Ozcan-Top and Demirörs, 2013; Schweigert et al., 2012). They are built over agile values and the improvement paths they suggest consider sustaining agility in the highest maturity levels. Two issues linger, though: the first is that they prescribe the practices the team should implement, even agile teams, which consider their work as a highly context-specific job to be prescribed (Fontana et al., 2014; Kettunen, 2012; Schweigert et al., 2012; Sidky et al., 2007); and the second is that the models still differ in their proposals, which indicates that the path to maturing in agile software development has not been uncovered yet.

These two issues thus, motivated this study. Our objective was to identify how agile software development teams evolve to maturity. We conducted an empirical research, through a multiple-case study approach, that identified how real agile teams evolve their practices and mature over time. The findings interest researchers, as they innovate in the underlying theory for a maturity model; and practitioners, as they provide practical guidelines for improving agile methods.

* Corresponding author at: Federal University of Paraná (UFPR), R. Dr. Alcides Vieira Arcoverde, 1225, Jd. das Américas, 81520-260 Curitiba, PR, Brazil. Tel.: +55 4133614904.
E-mail addresses: rafaela.fontana@ufpr.br (R.M. Fontana), victormeyerjr@gmail.com (V. Meyer), sheila.reinehr@pucpr.br (S. Reinehr), malu@ppgia.pucpr.br (A. Malucelli).

http://dx.doi.org/10.1016/j.jss.2014.12.032
0164-1212/

This paper is organized as follows: Section 2 outlines related work; Section 3 discusses our theoretical foundation and the theoretical framework we used for data analysis; and Section 4 presents the research structure. The results of the multiple-case study are presented in Section 5 and, finally, the findings are discussed and concluded in Sections 6 and 7, respectively.

2. Related work

Improvement paths in software engineering are currently defined by the guidelines given by CMMI-DEV and the international standard ISO/IEC 15504. The standard defines that software process improvement is accomplished through implementation of processes that address software acquisition, supply, engineering, operation, management, improvement, resources and infrastructure, reuse, configuration control, quality assurance and product quality (ISO/IEC, 2004).

Besides the incremental implementation of such processes, the capability to execute the processes should follow an evolutionary path. In ISO/IEC 15504 this capability is defined in terms of six levels: incomplete process, performed process, managed process, established process, predictable process and, lastly, optimizing process (ISO/IEC, 2004).

CMMI-DEV defines similar capability levels and, in addition, describes maturity levels to be followed for process improvement (CMMI Product Team, 2010). These levels are defined in terms of a set of processes that should be implemented and by the capability in which these processes should be performed. The first maturity level is named Initial, as processes here are usually ad hoc and chaotic; the second is called Managed, characterized by processes that lead projects to be performed and managed according to their documented plans. Next, comes third stage, Defined, in which processes are well characterized, understood and standardized in the organization. In fourth stage, "the organization and projects establish quantitative objectives for quality and process performance and use them as criteria in managing projects" (CMMI Product Team, 2010, p. 28) and for this reason it is called Quantitatively Managed. The highest maturity level is the Optimizing, in which improvement of the processes is continuous and based on a quantitative understanding of objectives and needs.

In the same path of CMMI two other models were created by Watts Humphrey: Personal Software Process (PSP) and Team Software Process (TSP). The first focuses on software individual discipline and the latter focuses on a disciplined group formed by PSP practitioners (Humphrey, 1995). The main difficulty that one faces when trying to use PSP is the amount of self-discipline that is required to fully apply the method. The first challenge is the need to record every single activity that the software engineer performs, the time that was spent in the activity and all the breaks taken during the work. This approach creates a bureaucracy in daily work that does not match agile approaches emphasis on people and interaction. PSP is also based on a seven-step roadmap that leads developer from a complete immature state (PSP0 – Baseline Personal Process) to a full mature state (PSP3 – Cyclic Personal Process). Similarly, TSP is a framework to be used in teams composed by PSP trained members (Humphrey, 2010). The model is based on an eight-step cycle: launch, strategy, plan, requirements, design, implementation, test and postmortem. The difficulty with TSP is the same of the PSP: the bureaucratic and prescriptive way to evolve work processes.

In the field of agile software development, two main lines of research have been studying maturity. The first focuses on adapting agile practices and principles to fit current software maturity models, such as CMMI-DEV. The second focuses on creating maturity paths related to agile software development values.

Since researchers and practitioners consider agile methods and software process improvement models as means to get the best from software development, there has been an increasing interest in combining both approaches. Starting with Mark Paulk explaining how Extreme Programming could be complementary to CMM (Paulk, 2001), a number of studies have either reported how companies have combined agile methods to CMMI-DEV requirements, or proposed new approaches to perform this combination (Al-Tarawneh et al., 2011; Anderson, 2005; Baker, 2006; Caffery et al., 2008; Cohan and Glazer, 2009; Jakobsen and Johnson, 2008; Lina and Dan, 2012; Lukasiewicz and Miler, 2012; Sutherland et al., 2007; Spoelstra et al., 2011; Tuan and Thang, 2013).

All these approaches result in adapting agile methods to fit assessment requirements in CMMI-DEV. They also accept that agile methods do not fit higher maturity levels, as the quantitative control of processes does not apply to agility (Lukasiewicz and Miler, 2012; Paulk, 2001). However, these initiatives recognize the value of having a combination of agility and disciplined processes (Boehm and Turner, 2004).

The second group of studies considers maturity in agile software development by keeping its focus on agility. The first agile maturity model we found published in the literature was the one by Nawrocki et al. (2001). They present a 4-level maturity model for XP (Extreme Programming). The authors' motivation was to identify if a specific organization is using XP or not. Typically, there are assessment problems because XP practices are not fully applied all the time. Thus, they say that a maturity model could be used as a reference point. For them, the model has to be hierarchical and identify practices for each level, such as CMMI-DEV.

The proposed model is called XPMM (eXtreme Programming Maturity Model) and was created based on intersections between XP and CMMI-DEV. The first level, named "Not compliant at all" means that the team applies no or a few XP practices. The second, "Initial", focuses on project teams and defines two process areas: Customer Relationship Management and Product Quality Assurance. The third, "Advanced" is focused on coding and, thus, the only process area is Pair Programming. The last level, called "Mature", has process areas related to the satisfaction of customers and developers. Here, the only process area is Project Performance. To be assigned to a specific level, the team has to follow the practices in this level and in the previous ones.

The authors assume that assessment should not be based on rich process documentation. They also state that many XP practices (e.g. simplicity, no functionality is added early) are difficult to assess. Then, they propose conversation and observation as assessment methods.

Another proposal for XP maturity model is the one presented by Lui and Chan (2005). They report a roadmap to aid the adoption of XP in Chinese inexperienced teams. They have analyzed the dependencies among XP practices and mapped them into a matrix. With the help of a visual data mining tool, they identified the ideal sequence for practices implementation.

They arrived at a four-stage XP implementation road map. In stage 1, the team should implement testing, simple design, refactoring and coding standard. In stage 2, the team should focus on continuous integration. In stage 3, the team should implement pair programming and collective ownership and, finally, in stage 4, the remaining XP practices would be adopted: metaphor, 40-h week, small release, on-site customer and planning game.

The model presented by Packlick (2007) is based on a single experience in Sabre Airline Solutions. He describes a different approach, based on the observation of real teams. The proposal is to use a goal-oriented approach because the author has observed that teams got more motivated to find out their own ways to get the job done. The AGILE Roadmap comprises five maturity levels that represent different learning stages an agile team should accomplish.

The goals are related to the AGILE acronym: Acceptance criteria; Green-bar tests and builds; Iterative planning; Learning and adapting; and, Engineering excellence. Each goal is detailed as a user's story