Expert Systems with Applications 42 (2015) 437–450

Contents lists available at ScienceDirect

Expert Systems with Applications

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

A procedure to detect problems of processes in software development projects using Bayesian networks

Mirko Perkusich [a,*], Gustavo Soares [a], Hyggo Almeida [a], Angelo Perkusich [b]

[a] Department of Computing and Systems, Federal University of Campina Grande, Rua Aprigio Veloso, 882, Bodocongo, 58109 900 Campina Grande, PB, Brazil
[b] Department of Electrical Engineering, Federal University of Campina Grande, Rua Aprigio Veloso, 882, Bodocongo, 58109 900 Campina Grande, PB, Brazil

ARTICLE INFO

Article history:
Available online 20 August 2014

Keywords:
Software process simulation modeling
Bayesian networks
Software process management
Software development project

ABSTRACT

There are several software process models and methodologies such as waterfall, spiral and agile. Even so, the rate of successful software development projects is low. Since software is the major output of software processes, increasing software process management quality should increase the project's chances of success. Organizations have invested to adapt software processes to their environments and the characteristics of projects to improve the productivity and quality of the products. In this paper, we present a procedure to detect problems of processes in software development projects using *Bayesian networks*. The procedure was successfully applied to *Scrum*-based software development projects. The research results should encourage the usage of *Bayesian networks* to manage software processes and increase the rate of successful software development projects.

1. Introduction

Over the years, software processes have evolved to take advantage of the organizations' structure and capabilities of human resources as well as the specific characteristics of the systems that they develop. There are several software processes models and methodologies such as waterfall, spiral and agile that are best suited depending on the project's and organizations' characteristics. For example, some would recommend waterfall for organizations with defined processes and large software development projects for a system with high criticality such as an online banking system. On the other hand, others would recommend agile for a project with a complex user interface and low criticality such as a social network application.

Even though software processes have evolved, there is still a low rate of successful software projects. According to a study performed in 2008 (Emam & Koru, 2008), only between 46 and 55 percent of IT projects succeed. For an applied discipline, this is a low success rate. Since software is the major output of software processes and software development projects, they are both correlated. Increasing the software processes management quality should increase the software development project's chances of success.

One approach to assess and manage software processes is through software process simulation modeling (Kellner, Madachy, & Raffo, 1999). In these approach, researchers construct models that represent software processes to assist management and increase the project's chances of success. With this purpose, model developers use techniques such as *system dynamics*, discrete events simulation and *Bayesian networks* (Zhang, Kitchenham, & Pfahl, 2008). Even though this approach has the potential to encompass key factors of software process models such as specification, quality and development (Sommerville, 2010), in most cases, researchers have used it for a limited scope of processes. For instance, Abouelela and Benedicenti (2010) and Jeet, Bhatia, and Minhas (2011a) applied this approach only for quality management. Furthermore, most studies do not present a procedure to use the model. A procedure to construct and use models to assist on software process management is essential to give model developers a common knowledge and instructions to optimize the chances of constructing a model that best suits the project that it will be applied to.

By modelling software development processes key factors, it is possible to assist on continuous improvement by detecting their problems. In this paper, as shown in Section 3, we present a procedure to construct and use *Bayesian networks* for this purpose. The proposed procedure can be applied to detect problems in any software development methodology or framework.

We used *Bayesian networks* due to their capability to handle uncertainty and also because of their ease of understanding and

* Corresponding author.
E-mail addresses: mirko.perkusich@copin.ufcg.edu.br (M. Perkusich), gsoares@computacao.ufcg.edu.br (G. Soares), hyggo@dsc.ufcg.edu.br (H. Almeida), perkusic@dee.ufcg.edu.br (A. Perkusich).

http://dx.doi.org/10.1016/j.eswa.2014.08.015
0957-4174/

modification by practitioners. Researchers have applied this technique for expert systems in areas such as software maintenance project management (de Melo & Sanchez, 2008), safety control in complex project environments (Zhang, Wu, Ding, Skibniewski, & Yan, 2013) and performance forecast of innovation projects (de Oliveira, Possamai, Dalla Valentina, & Flesch, 2012).

To validate the procedure, we applied it to *Scrum*-based software development projects. We chose *Scrum* because it is the most popular agile framework (VersionOne, 2013), has an active community and specialized organizations that support its state of art such as *Scrum Alliance* and *Scrum.org*. Our validation reasoning is similar to Lee, Park, and Shin (2009), in which the authors present a procedure to construct and use *Bayesian network* for risk management in large engineering projects and validate it by applying it to a specific purpose: the Korean shipbuilding industry. We divided the procedure validation into two steps: (i) validate the *Bayesian network* and (ii) its usage process. To validate the *Bayesian network*, as shown in Section 4.6.1, we individually tested its node probability tables and used simulated scenarios to test its outputs. To validate the *Bayesian network* usage process, as shown in Section 4.6.2, we performed a case study for two real projects in a company located in Brazil. For both cases, it helped to improve the quality of the processes thereby improving the project's chances of success with positive cost-benefit.

This paper is organized as follows: in Section 2, we present relevant literature review focusing on software processes evolution, software process simulation modelling, *Bayesian networks* and *Scrum*; in Section 3, we present the procedure in details and a guideline to build *Bayesian networks* for software processes; in Section 4, we present an overview of the procedure's application to *Scrum*-based software projects, which Perkusich, de Almeida, and Perkusich (2013a) presents in detail, and information regarding its validation which encompassed the *Bayesian network* and its usage process; and, in Section 5, we present our conclusions, current limitations and future works.

2. Literature review

2.1. Software processes

According to Sommerville (2010), software processes are interleaved sequences of technical, collaborative, and managerial activities with the overall goal of specifying, designing, implementing, and testing a software system. Furthermore, they can be described with roles, products, and pre- and post-conditions. According to Boehm (1988), they provide guidance on the order in which a project should carry out its major tasks. Over the years, software processes have evolved to handle the expectations of the software development projects in the best way possible. As a consequence, several software process models and methodologies were proposed in an attempt to increase the chances of success of the software development projects. Each of these had their advantages and disadvantages.

According to Boehm (2006), during the early years of software engineering in the 1950s, software processes were plan-driven and sequential because software projects were managed as hardware projects. During the 1960s, because software could be easily modified, many programmers started to use the "code and fix" approach and this created heavily patched spaghetti code.

In the 1970s, as a reaction to the problems caused by the "code and fix" approach, waterfall and formal methods were proposed. The waterfall process represents the fundamental process activities as process phases (Royce et al., 1970). It was intended to be iterative but it was interpreted as sequential. Formal methods are mathematical techniques, often supported by tools, for developing software and hardware systems. Mathematical rigor enables users to analyze and verify these models at any part of the program life-cycle. These parts are: requirements engineering, specification, architecture, design, implementation, testing, maintenance, and evolution (Woodcock, Larsen, Bicarregui, & Fitzgerald, 2009). As a reaction to these heavy-weight processes, the Rapid Application Development (RAD), which is incremental and iterative (Martin, 1991), was proposed.

Later in the 1980s, software process standards were proposed to avoid process noncompliance and Software Capability Maturity Models were used to assess an organization's software process maturity. The Software Capability Maturity Model (SW-CMM) and ISO-9001 were created and largely used. The SW-CMM provides an effective framework for both capability assessment and improvement (Humphrey, 1989). ISO-9001 is part of the ISO-9000 family of standards that is related to quality management systems. It is not specific for software development, but it was largely used for external quality assurance of software development projects. In 1987, Osterweil (1987) proposed the usage of programming techniques and formalisms to express software process descriptions. Furthermore, during the 1980s new software development processes were proposed such as evolutionary (McCracken & Jackson, 1982), *Cleanroom* (Mills, Dyer, & Linger, 1987) and risk-driven spiral (Boehm, 1988).

During the 1990s, due to the need to reduce time-to-market, a major shift occurred away from sequential models towards agile methods such as *Adaptive Software Development* (ASD) (Highsmith, 1999), *Crystal* (Cockburn, 2001), *Dynamic Systems Development* (DSDM) (Stapleton & Constable, 1997), *eXtreme Programming* (XP) (Beck, 2000), *Feature Driven Development* (FDD) (Coad, de Luca, & Lefebvre, 1999), Kanban (Anderson, 2010), *Scrum* (Cohn, 2009) and *Scrumban* (Ladas, 2009). This approach relies on lightweight processes with an incremental approach to software specification, development, and delivery to maximize value delivery to the customers. It intends to deliver working software quickly to users, who can then propose new and changed requirements to be included in later iterations of the system (Sommerville, 2010). On the other hand, even though lightweight (agile) software processes arose, heavyweight (plan-driven) software processes were still used and new ones were proposed in the late 1990s and the following years such as the *Unified Software Development Process* (Jacobson, Booch, & Rumbaugh, 1999) and the *Rational Unified Process* (RUP) Kruchten (2003). In general, the choice of the best suited software process for a project depends on the type of product, size of the project, and the business requirements. According to Sommerville (2010), agile methods are best suited for small or medium-sized projects with low criticality. Plan-driven methods are best suited for large companies with defined processes and large projects.

Software companies have paid greater attention as to improve process productivity and quality of the delivered products. As a consequence, the evaluation of software processes became a very important issue because software is its major outcome (Li, Li, Wu, & Song, 2012). Software processes can be improved by process standardization. This leads to improved communication, a reduction in training time, and also makes automated process support more economical.

2.2. Software process simulation modeling

Another way to improve software processes is through software process simulation modeling because it can support software process management (Kellner et al., 1999). The goal of software process simulation modeling is to use technologies, people, and tools to collaboratively increase the chances of success of software development projects. These models may focus on development