Available online at www.sciencedirect.com

ScienceDirect

International Journal of Project Management 35 (2017) 13–27

International Journal of Project Management

www.elsevier.com/locate/ijproman

Excessive software development: Practices and penalties

Ofira Shmueli [a], Boaz Ronen [b,*]

[a] Ben-Gurion University of the Negev, Industrial Engineering and Management Department, Israel
[b] Tel Aviv University, Faculty of Management, Israel

Received 19 October 2015; received in revised form 25 September 2016; accepted 3 October 2016
Available online 28 October 2016

Abstract

This study focuses on the tendency to develop software excessively, above and beyond need or available development resources. The literature pays little attention to this issue, overlooking its crucial impact and penalties. Terms used in reference to excessive software development practices include over-requirement, over-specification, over-design, gold-plating, bells-and-whistles, feature creep, scope creep, requirements creep, featuritis, scope overload and over-scoping. Some of these terms share the same meaning, some overlap, some refer to the development phase, and some to the final system. Via a systematic literature search, we first demonstrate the poor state of research about excessive software development practices in the information systems and project management areas. Then, we suggest a framework consolidating the problems associated with excessive software development in three 'beyond' categories (*beyond needs*, *beyond resources*, *beyond plans*), describe and analyze their causes, consequences, boundaries and overlapping zones. Finally, we discuss the findings and present directions for future research.

Keywords: Software development; Project management; Over-requirement; Over-specification; Over-design; Gold-plating; Bells-and-whistles; Mission creep; Feature creep; Scope creep; Requirements creep; Featuritis; Scope overload; Over-scoping

1. Introduction

Over four decades ago, Brooks (1975) observed that the most difficult part of developing a software system is deciding precisely what to build. He further noted that, if done incorrectly, no other part of the development work cripples the resulting system as much or is more difficult to undo later. This observation, which has been repeatedly acknowledged over the years by academic research and practical experience, still holds today.

Catering to user, market or organizational needs[1] in software development projects has a major impact on project success. Not meeting just the right needs is one of the reasons for project failure (Charette, 2005; Keil et al., 1998), perhaps even the most crucial one (Kliem, 2000; Longstaff et al., 2000). Software scoping is a critical project process (Zwikael and Smyrk, 2011) and project success is sensitive to the defined scope (Cano and Lidón, 2011). Although wrong scope definition refers to both scoping under and over the actual needs (Bjarnason et al., 2012; Buschmann, 2009; Zwikael and Smyrk, 2011), excessively loading the scope is much more common (Bjarnason et al., 2012; Boehm, 2006; Karlsson et al., 2007), and thus stands at the focus of this work. Exceeding the right scope expands project size, which is a major risk dimension in software development projects (McFarlan, 1981; Zmud, 1980) in the sense that project risk is an increasing function of project size (Barki et al., 1993; Glass, 1998; Houston et al., 2001; Maguire, 2002). In comparison to smaller projects, large-scale projects fail three to five times more often (Charette, 2005), are much more prone to unexpected colossal events including even bringing an organization down (Flyvbjerg and Budzier, 2011), and have a 65% probability of being stopped and abandoned (Jones, 2007).

This study relates mainly to traditional plan-based software development methodologies, such as the waterfall approach. Current agile techniques claim to resolve problems associated

* Corresponding author.
E-mail addresses: ofirash@post.bgu.ac.il (O. Shmueli), boazr@post.tau.ac.il (B. Ronen).

[1] See Appendix A for explanations of the software engineering terms.

http://dx.doi.org/10.1016/j.ijproman.2016.10.002
0263-7863/00/

with plan-based methodologies but the debate on which methodology is more effective (Beck and Boehm, 2003; DeMarco and Boehm, 2002), especially in the requirements engineering (RE) context still exists (Dyba and Dingsyr, 2009; Inayat et al., 2015a). Critics of agile techniques claim that agile requirements engineering concepts lead to neglecting non-functional requirements related to performance, security, and architecture (Cao and Ramesh, 2008; Dyba and Dingsyr, 2008; Maiden and Jones, 2010). Although studies that describe requirements engineering practices in an agile context address some of these problems (Bakalova and Daneva, 2011; Lucia and Qusef, 2010), knowledge about the solutions that agile brings to RE is fragmented and whether while introducing solutions to these problems new challenges are introduced is yet to be examined (Inayat et al., 2015a). Accordingly, recent studies claim that this field is still immature and needs further research on agile RE and its real-world impact and applications (Maiden and Jones, 2010; Inayat et al., 2015a, 2015b). However, the understanding that *no size fits all* (Boehm and Lane, 2010a) and that both approaches have their merits and excel under appropriate conditions, suggests more balanced hybrid approaches that integrate both into the right mix for each specific project (Boehm and Turner, 2003, 2005; Boehm et al., 2010; Dyba and Dingsyr, 2008, 2009).

The risky practice of expanding a software project to include excessive functionality and capabilities[2] is referred to in the literature by a variety of partially overlapping terms, including: over-requirement, over-specification, over-design, gold-plating, bells-and-whistles, mission creep, feature creep, scope creep, requirements creep, featuritis, scope overload and over-scoping.

While all these terms relate overall to excessive software development practices, as elaborated upon in the next sections, there are some differences, depending, for example, on the project development phase in which each practice takes place, whether requirements[3] added under the excessive development practice can be implemented within the project constraints or not and whether the added requirements are essential, just optional or completely unnecessary. However, once extra features are introduced into a project excessively, they are seldom eliminated regardless of necessity or of how and during which project phase they are included within the scope (Dominus, 2006; Wetherbe, 1991).

Excessive software development practices are considered risky practices. They impose a variety of penalties on project outcome, with many negative consequences on project schedule, quality and costs (Bernstein, 2012; Bjarnason et al., 2010; Buschmann, 2009, 2010; Coman and Ronen, 2010; Ronen et al., 2012). While some studies refer to a change in requirements of about 25% on average (Jones, 1994; McConnell, 1996), others present an average total volume growth of 14% to 25% for software projects in various domains, with a monthly rate of change in requirements of 1% to 3.5% (Choi and Bae, 2009; Jones, 1996). Jones (1996), however, emphasizes that these numbers can be misleading since the maximum growth rate observed in many cases exceeded 100%. Coman and Ronen (2009b) ascribe over 30% of the features in financial software applications serving such organizations as banks or insurance companies to excessive software development. They claim that over 25% of the software development efforts in R&D organizations are devoted to issues and activities that do not add value (Coman and Ronen, 2010). Considering the conservative estimate of 25% superfluous scope (Battles et al., 1996; Coman and Ronen, 2010), one must wonder what the costs of excessive software development amount to in terms of budget and schedule overruns as well as damage to system quality and integrity. According to McConnell (1996), due to the multiplicative costs associated with doing work downstream, these costs probably amount to much more than 25%. Using the COCOMO II estimation model (Boehm et al., 2000a, 2000b), which considers the exponential nature of the development effort, the estimated cost increment might indeed be even higher than the scope increment, at least with respect to the development activity. Non-development project activities, such as preparing infrastructures or training users, are affected by size and content and are expanded as well due to excessive software development practices. To show that costs can be reduced by eliminating excess, Battles et al. (1996) provide an example of an electric utility which succeeded in reducing the software development budget by 30% without reducing performance by avoiding unnecessary upgrades and non-critical work. Ronen et al. (2012) refer to a cellular phone service provider that by adopting the 25/25 rule in software development, i.e., terminating 25% of the projects and eliminating 25% of the features[4] in the remaining projects, improved the project completion rate and development pace.

Although it is thus extremely important to explore the risky excessive software development practices, enhance the knowledge and awareness of them, and to recommend remedies for their mitigation, a literature search reveals only thin, spare, fragmented and scattered research on these issues. This work makes three main contributions to this challenge. First, via a systematic literature search, focused on title, abstract and keywords of articles in top-rated journals, it unravels the small amount of current relevant research in these leading journals. Second, it gathers and elaborates upon the different terms associated with excessive software development practices and provides a comprehensive picture regarding their nature, causes and consequences. Third, based on the findings and analysis presented here it proposes a research agenda for future research in several directions.

The rest of this paper is dedicated to reviewing the various excessive software development practices and to exposing the current poor state of relevant research. Section 2, first identifies the various terms that relate to excessive software development practices and then presents the findings of a systematic literature search for relevant research. Section 3 consolidates the various excessive software development practices in three 'beyond' categories, and analyzes their nature, causes, and boundaries. Finally, Section 4 discusses the findings, conclusions and implications and proposes a research agenda. A glossary of basic software engineering terms used here is

[2] See Appendix A for explanations of the software engineering terms.
[3] See Appendix A for explanations of the software engineering terms.
[4] See Appendix A for explanations of the software engineering terms.