Exploring the Relationship between Perceptions of Agile Software Development and Technical Debt

Emergent Research Forum Paper

Corey Baham
Oklahoma State University
corey.baham@okstate.edu

Abstract

In many fast-paced software development environments, there is a push to produce working software quickly in order to meet timely deadlines and fulfill project objectives. Currently, there is host of research on the positive effects of agile development use on software development project outcomes; however, many of the long-term consequences have not been fully examined. The concept of technical debt presents an opportunity to conceptualize and study such consequences. The purpose of this study is to explore the relationship between perceptions of agile software development methods and the accumulation of technical debt. We use preliminary interviews to learn about how developers understand this relationship. We then use these insights to design a field study to gain further insight into the impact of software development methods on technical debt.

Keywords

Technical Debt; Agile Use; Agile Project Management; Agile Teams

Introduction

Agile software development (ASD) method represent a departure from the heavily regimented and document-driven procedures of traditional, waterfall approaches. In contrast, ASD methods focus on adapting quickly to changing user requirements and to using less time for documentation in order to build working software quickly and iteratively through a collaborative effort. Prior research highlights several benefits of using ASD methods over traditional software development (SD) methods such as faster time to market, higher software quality, and higher customer satisfaction (Overhage and Schlauderer 2012). One important, yet often overlooked outcome variable is what practitioners term “technical debt”. Technical debt describes the visible and invisible results of past decisions and short-term compromises about a software that creates complexities and could negatively affects its future (Krutchten et al. 2012; Elbanna and Sarker 2016). Extant literature suggests that technical debt explains how inefficiencies in managing the SD process may accumulate over time and stymie the ability to update code efficiently (Sterling 2010). Consequently, the dependency on brittle code may hinder a development team’s ability to respond quickly to changing user requirements. Although ASD methods aim to provide “continuous attention to technical excellence” (Beck et al. 2001), one might argue that the rapid development practices of ASD methods might facilitate rather than reduce technical debt.

The purpose of this study is to examine the relationship between software team perspectives on adopting ASD methods and technical debt, particularly as it pertains to the need to reduce development time. To accomplish this objective, first we conducted preliminary interviews with SD professionals to better understand the proposed relationship. Second, we seek to design a field study to sample team members of SD projects across various team sizes, industries, project sizes, and delivery methods. The rest of the paper is organized as follows: The following section discusses the literature in technical debt. Next, the methodology, expected contributions, and future research are discussed.
Literature Review

Agile Software Development

We define the practice of agile software development as a software team’s ability to respond to changing user requirements through a process of continual readiness (Conboy 2009; Lee and Xia 2010). Through agile practices, software teams may quickly and inherently create, embrace, and learn from change while contributing to perceived customer value (Conboy 2009).

Extant research indicates that the concept of agility first appeared in the mainstream business literature in the early 1990s (Goldman et al. 1991). Prior studies explored the concept of agility concerning manufacturing, management, product development, and other business research development (Takeuchi and Nonaka 1986; Sugimori 1977). Despite the contributions from these fields, the term “agile” became widely popular after the advent of the Manifesto for Agile Software Development (Beck et al. 2001) in 2001, a document developed by group of software development practitioners that marked a new approach to building software.

Although the agile philosophy represents lean, lightweight development methods, it does not completely abandon documentation, contracts, processes, tools, and plans; rather, it places more emphasis on the people who are involved and on creating working software (Beck et al. 2001). Prior research identifies several benefits of agile method in completing organizational projects including adaptability, flexibility, and project visibility (Kenefick 2011). Overviews of ASD describe a number of method that subscribe to the principles of the Manifesto for Agile Software Development including Scrum and Kanban (Abrahamsson et al. 2002; Wester 2014). Extant literature elaborates on the merit of these methodological frameworks for delivering project requirements using adaptive and agile processes (Baskerville et al. 2003, Daneva et al. 2013).

Technical Debt

Technical debt is a term borrowed from financial debt and introduced by software developer, Ward Cunningham in 1992, who described it as the activity of writing “not quite right code” in order to reduce the time to market of a piece of software (Cunningham 1992). Since then, technical debt has been defined many different ways with common conceptualizations referring to the tradeoff between short-term decisions and long-term consequences (Lim, Taksande, and Seaman 2012; McConnell 2007). In this study, we refer to technical debt as the visible and invisible results of past decisions and short-term compromises about a software that creates complexities and could negatively affects its future (Krutchen et al. 2012; Elbanna and Sarker 2016). With ASD methods, technical debt is often captured in a project backlog as work to be completed. Technical debt that is not repaid will keep on accumulating interest, making it hard to implement changes later on. Thus, unaddressed technical debt increases software entropy, leading to inevitable decline and degeneration.

The extant literature distinguishes between intentional and unintentional technical debt. Where intentional technical debt refers to debt that is initiated knowingly by decision makers who compromise some level of software completeness often to satisfy competing business interests, unintentional technical debt refers to resulting the technical debt that is accumulated unknowingly (McConnell 2007). Additionally, the extant literature identifies different types of technical debt such as software debt, design debt, quality debt, testing debt, and configuration management debt (Sterling 2010). Interestingly, intentional technical debt has been linked to decisions to incur software debt for short-term, tactical reasons (Ashmore and Runyan 2015). Despite having a more long-term view of the software, developers tend to be more aware of the presence and possible difficulties of technical debt (Lim, Taksande, and Seaman 2012), but submit to the desires of managers.

Technical Debt

ASD has been touted for its ability to delivery working software faster and with less defects than traditional methods. In fact, empirical findings support the idea that the use of ASD produces faster times to market, higher software quality, and higher customer satisfaction (Cao et al. 2009; Overhage and
Schlauderer 2012). However, many of these studies examine the success of one or more ASD projects within a short time span. The literature is less forthcoming concerning how to maintain ASD in the long-term. Consequently, since technical debt is predicated on the tradeoff between short decisions and long-term consequences, the lack of ASD studies beyond the pilot stages has created a gap in our ability to understand the impacts of ASD on software quality over time. On one hand, the collaborative work processes and iterative development cycles in ASD may provide frequent opportunities to improve software quality. The close interaction between business stakeholders and the development team as part of ASD may help to mitigate some of the miscommunications identified with traditional software development (Elbanna and Sarker 2016). Prior studies highlight the benefits of specific ASD practices and processes such as iteration planning on technical debt management (Lim, Taksande, and Seaman 2012; Holvitie et al. 2014).

On the other hand, the pressure of producing software more rapidly may lead to more shortcuts and less quality than alternative delivery methods. Prior research suggests that rushed coding jobs lead to the accumulation of technical debt (Cohn 2009). In such scenarios, how ASD is implemented may impact software quality. For instance, the need to significantly reduce development time, adhere to strict time-boxing, and deliver functional requirements for business use may lead to pressuring a software team to release software quicker to satisfy immediate managerial and customer requirements. Consequently, technical debt can accumulate over time and lead to unexpected project delays (Elbanna and Sarker 2016).

Although many champions of ASD (e.g. Martin Fowler, etc.) warn of the consequences of technical debt, little research has been conducted to show whether perspectives on how and why ASD is implemented have an impact on technical debt. In this study, we examine software team member perspectives on ASD as a perceived way to deliver a rushed coding job and its effects on three dimensions of software debt, namely code cleanliness, code documentation, and code optimization. Code cleanliness refers to the degree to which code has been refactored so that extraneous code is removed and it complies with other pattern structures that may exist in the product (Cohn 2009). Clean code can be understood as code that is perceived to be well-written and that the team does not feel the need to immediately refactor or rewrite it (Cohn 2013). Code documentation is documentation that accompanies software that explains how it operates. This documentation includes contents in the source code that allow a software team members to quickly understand otherwise arbitrary code. Code optimization refers to methods of code editing to improve code quality and the efficiency. Code optimization may involve reducing overall code size, memory consumption, and the number of inputs and outputs operations (Sedgewick 1984).

**Research Methodology**

To gain insight into the relationship between ASD and technical debt, we use a mixed methods approach. First, we conduct interviews with 40 practitioners with experience using ASD in order to gain insight into the processing of accumulating technical debt in SD projects. These practitioners included developers, quality assurance testers, and Scrum Masters. Each practitioner was asked about the presence of technical debt in their most recent ASD project, including the sources, consequences, and steps to resolve technical debt. Interviewees were given the same definition of technical debt and further explanation was given at the beginning of the interview session as needed. The results of our preliminary interviews demonstrate two different perspectives concerning the relationship between agile method and technical debt. One perspective is that agile methods improve software quality, thereby reducing technical debt (in the form of code cleanliness and code documentation). One developer described the visibility of technical debt between waterfall and agile method as follows,

“In waterfall, [technical debt] could have easily been forgotten about, so unless somebody said, “oh wait, don’t forget you have to go clean that up.” I think agile was more clear about it.”

Another perspective is that agile methods may lead to technical debt as software quality may be compromised when rapid development is emphasized. One developer explained,
“Since our [agile] project is time-bound, we do not have the time to go back and clean. Sometimes while working you know that you are creating technical debt, but you do not have enough time to do it the right way.”

These differing perspectives motivated the second phase of our study in which we seek to answer the following question: How do software team member perspectives on the motivation for adopting ASD methods (e.g. perceptions of ASD being implemented to rush coding jobs) effect technical debt? Based on the differing perspectives noted above, we formulate our hypotheses as follows:

H1: Software teams that interpret adopting agile as a way to deliver a rushed coding job will tend to increase technical debt.
H2: The lack of perceived need for clean code will positively moderate a software team’s interpretation that adopting agile is a way to deliver a rushed coding job, which will tend to increase technical debt.

Second, we seek to clarify the relationship between ASD and technical debt by conducting a field study of agile team members across multiple contexts, taking into consideration variables such as project size, team size, project scope, and agile method. This study intends to provide a rich understanding of perspectives on ASD and its possible effects on technical debt. We will test the impact of adopting agile on perceptions of implementing ASD to rush coding jobs, and in turn, its impact of technical debt. Additionally, we will test the perceived lack of need for clean code as a moderator of adopting agile and perceptions of a rushed job as shown in Figure 1.

---

**Data collection and analysis**

This research will utilize an online survey for data collection. This process is projected to begin in July 2017 and continue to September 2017. We will modify existing scales for adopting agile from prior research (Maruping et al. 2009) and develop novel scales for the remaining constructs to test our model. The data will be analyzed using PLS.

**Expected Contributions and Future Research**

This research makes a number of contributions to both research and practice. First, this study will test software team perceptions on three dimensions of intentional technical debt, namely code cleanliness, code documentation, and code optimization. Researchers and practitioners need to focus on antecedent to specific types and dimensions of technical debt in order to deepen our understanding. Next steps include refining our construct definitions, developing, and validating our scales before we deploy our survey. Second, empirical investigation of the relationship between ASD and technical debt will help provide a deeper theoretical rationale behind ASD methods (Rathor et al. 2016). Finally, this study will explain the role of short-term decision on long-term consequences, as prior research calls for a deeper understanding of ASD post-implementation (Abrahamsson et al. 2009).
References


Cohn, M. 2009. Succeeding with agile using scrum.


Sedgewick, R. 1984 *Algorithms. Series in Computer Science.* Addison-Wesley, USA.


