Association for Information Systems

AIS Electronic Library (AISeL)

International Research Workshop on IT Project Management 2018

International Research Workshop on IT Project Management (IRWITPM)

Winter 12-13-2018

Leveraging the Planning Fallacy to Manage Technical Debt in Agile Software Development Projects

Maheshwar Boodraj

Georgia State University J Mack Robinson College of Business, mboodraj@boisestate.edu

Follow this and additional works at: https://aisel.aisnet.org/irwitpm2018

Recommended Citation

Boodraj, Maheshwar, "Leveraging the Planning Fallacy to Manage Technical Debt in Agile Software Development Projects" (2018). *International Research Workshop on IT Project Management 2018*. 2.
https://aisel.aisnet.org/irwitpm2018/2

This material is brought to you by the International Research Workshop on IT Project Management (IRWITPM) at AIS Electronic Library (AISeL). It has been accepted for inclusion in International Research Workshop on IT Project Management 2018 by an authorized administrator of AIS Electronic Library (AISeL). For more information, please contact elibrary@aisnet.org.

Leveraging the Planning Fallacy to Manage Technical Debt in Agile Software Development Projects

Maheshwar Boodraj
Georgia State University
mboodraj1@gsu.edu

ABSTRACT

One of the primary reasons for using agile software development (ASD) methods is to be agile – to deliver working software quickly. Unfortunately, this pressure often encourages ASD practitioners to make long-term trade-offs for short-term gains (i.e., to accumulate technical debt). Technical debt is a real and significant business challenge. Indeed, a recent study provides a conservative estimate of $361,000 of technical debt for every 100,000 lines of code. In this study, I examine the impact of the planning fallacy – people's tendency to underestimate the time required to complete a project, even when they have considerable experience of past failures to live up to planned schedules – on the accumulation of technical debt in ASD projects. Using an experiment, I seek to establish a causal relationship between the planning fallacy and technical debt and to demonstrate that solutions to the planning fallacy can be leveraged to manage technical debt in ASD projects.

Keywords

Planning Fallacy, Technical Debt, Agile Software Development, Time Underestimation, Scope Overload, Code Debt, Decision Making, IT Project Management, Experimental Design.

INTRODUCTION

Many of today's innovations are driven by software. For example, autonomous cars, such as those being developed by Tesla, require sophisticated software to deliver control, navigation, monitoring, and entertainment capabilities (Archer Software, 2016b); conversational assistants such as Apple's Siri and Amazon's Alexa require advanced speech recognition algorithms to deliver a more natural user experience (Pisipati, 2017); and smart homes require software with ever-expanding capabilities to integrate and manage the multitude of smart devices that are being produced daily (Archer Software, 2016a). To meet the software demands to power these innovations, many organizations are turning to ASD methods such as Scrum and eXtreme Programming (XP).

ASD methods are attractive for several reasons. For example, they enable organizations to go to market faster, they enable project managers to develop products that better meet customer requirements, and they are more responsive to today's rapidly changing business environments. ASD methods are not without disadvantages, however. For example, to deliver software quickly, many ASD methods foster the accumulation of "technical debt" (Cunningham, 1992). Technical debt describes the debt that development teams incur when they opt for an easy or quick approach to implementing in the short term, but with a greater possibility of a negative long-term impact (Alves, Mendes, de Mendonça, Spínola, Shull and Seaman, 2016). Examples of technical debt include poorly written source code that may need to be refactored later, minimal documentation which may necessitate additional person-hours in the long-run to modify the source code, and limited testing which may manifest itself in poor quality software or degraded user experience over time.

Technical debt is a real and significant business challenge. This is underscored in a recent study by Curtis, Sappidi and Szynkarski (2012), who analyzed 365 million lines of code – from 745 business applications spanning 160 companies across ten industries – to provide a conservative estimate of $361,000 of technical debt for every 100,000 lines of code. While much of this technical debt is the result of deliberate choices by project managers and project teams, some of this technical debt is incurred inadvertently (Fowler, 2009; Lim, Taksande and Seaman, 2012). The present challenge, therefore, is to reduce technical debt when it is undesirable. Over the last several years, there has been increasing interest from both research and practice to understand and manage technical debt (Kruchten, Nord, Ozkaya and Falessi, 2013), which presents a timely opportunity for me to contribute to the current academic discourse on technical debt.

I seek to contribute to the literature on technical debt by introducing the planning fallacy as a useful theoretical lens for exploring time underestimation and its effects on technical debt in ASD projects. The planning fallacy refers to people's tendency to "underestimate the time required to complete a project, even when they have considerable experience of past failures to live up to planned schedules" (Kahneman and Tversky, 1979, 1982, p. 415). A familiar example of the planning fallacy can be observed when academics take home briefcases packed with work on Fridays, fully intending to complete everything over the weekend, despite not having gone beyond the first few tasks on any previous weekend (Buehler, Griffin and Ross, 1994).

By using a randomized lab experiment, I aim to establish a relationship between the planning fallacy and technical debt – a relationship which remains largely unexplored in the literature. In doing so, I hope to prompt both researchers and practitioners to leverage solutions to the planning fallacy as potential solutions to managing technical debt in ASD projects. This would represent a significant contribution to practice by helping to reduce the $1 trillion global technical debt bill (Gartner, 2010). It would also represent a significant contribution to research by empirically bridging the nomological networks of the planning fallacy and technical debt. I, therefore, seek to answer the following two research questions in this study:

RQ1: What is the impact of the planning fallacy on the accumulation of technical debt in ASD projects?

RQ2: How does the planning fallacy contribute to the accumulation of technical debt in ASD projects?

THEORETICAL BACKGROUND

Technical Debt

Technical debt, a term coined by Cunningham (1992), describes the debt that development teams incur when they opt for an easy or quick approach to implementing in the short term, but with a greater possibility of a negative long-term impact (Alves et al., 2016; Li, Avgeriou and Liang, 2015). In a useful metaphor, Ampatzoglou, Ampatzoglou, Chatzigeorgiou and Avgeriou (2015) likened technical debt to financial debt, which consists of a principal and interest. According to Camden (2013), paying back the principal involves implementing the correct replacement, while paying back the interest involves working around the incorrect implementation. The primary difference between technical debt and financial debt, however, is that the interest associated with technical debt may or may not need to be repaid (Guo, Spínola and Seaman, 2016). For example, technical debt may not need to be repaid if an organization creates a temporary system, which they intend to discard in a few months or a few years. A major problem occurs, however, when these temporary systems become permanent fixtures in organizations.

Several factors contribute to the accumulation of technical debt in ASD projects. For example, technical debt may be caused by an emphasis on quick delivery, architecture and design issues, inadequate test coverage, unclear understanding of system requirements, overlooked and delayed solutions and estimates, inadequate or delayed refactoring, and duplicate code (Behutiye, Rodríguez, Oivo and Tosun, 2017). One of the most common of these factors is an emphasis on quick delivery. Unmanaged, technical debt can lead to several undesired consequences. For example, technical debt can lead to reduced productivity, system quality degradation, increased cost of maintenance, complete redesign or rework of a system, market loss, and damaged business relationships (Behutiye et al., 2017). However, when technical debt is incurred strategically, it can provide benefits such as allowing organizations to capture market share or collect early customer feedback (Lim et al., 2012).

In a systematic literature review of 38 primary studies on technical debt in ASD projects, Behutiye et al. (2017) identified five major research areas, of which, managing technical debt in ASD – the focus of this study – generated the most interest. According to Kruchten et al. (2013, p. 51), a "better understanding of the concept of technical debt, and how to approach it, both from a theoretical and a practical perspective is necessary to advance its state of the art and practice." This call-to-action is further echoed by Alves et al. (2016, p. 118) who argued that "it is necessary to conduct further studies in the area to investigate new techniques and tools that could support developers with the control of technical debt." According to Behutiye et al. (2017), the most common strategies for managing technical debt in ASD projects include (1) the use of specific tools, approaches, and models, (2) refactoring – restructuring existing computer code without affecting its external behavior, and (3) enhancing the visibility of technical debt. This study focuses on a less explored, but extremely important, approach – improving estimation techniques. Specifically, I explore the role of the planning fallacy on the accumulation of technical debt in ASD projects.

The Planning Fallacy

The planning fallacy refers to people's tendency to "underestimate the time required to complete a project, even when they have considerable experience of past failures to live up to planned schedules" (Kahneman and Tversky, 1979, 1982, p. 415). The planning fallacy is, therefore, not only about making optimistic predictions but about doing so *despite* having historical evidence to the contrary. According to Lovallo and Kahneman (2003, p. 3), the planning fallacy causes managers to "make decisions based on delusional optimism rather than on a rational weighting of gains, losses, and probabilities." We can observe the planning fallacy in everyday situations such as when students underestimate the time it will take them to prepare for an exam or complete a research paper despite ample historical evidence to the contrary.

While several psychological mechanisms have been used to explain the planning fallacy (Buehler and Griffin, 2015), the main cognitive model is *the inside and outside view* proposed by Kahneman and Tversky (1979). The inside view focuses on *singular* information such as the details of the focal task, while the outside view focuses on *distributional* information such as how the current task fits into the set of related tasks (Buehler, Griffin and Peetz, 2010). In this model, the inside view is expected to lead to time underestimation whereas the outside view is expected to lead to more realistic, yet still imperfect, estimates (Kahneman and Lovallo, 1993). Also, this cognitive model is quite robust and has been demonstrated in both real-world examples and lab experiments (Buehler et al., 1994; Flyvbjerg, 2013; Kahneman and Lovallo, 1993).

In a recent study, Shmueli, Pliskin and Fink (2016) explored whether taking an outside-view approach, recommended for reducing the irrational behaviors associated with the planning fallacy, could also reduce time underestimation, scope overload, and over-requirement (also known as "gold plating") problems plaguing planning decisions in software development projects. In their study, Shmueli et al. (2016) found that these problems are mitigated, but not eliminated, by presenting reference information about past completion times and by having participants adopt a consultant role – both outside view mechanisms. Further, Shmueli et al. (2016) observed relatively high correlations among these three variables and suggested that the correlations may be indicative of causal relationships that exist among the three variables. In this study, I seek to extend the work of Shmueli et al. (2016) by testing one of these causal relationships (time underestimation → scope overload), which I discuss in the next section, and by exploring its potential contribution to the accumulation technical debt in ASD projects.

RESEARCH MODEL AND HYPOTHESES

Research Model

Based on the preceding discussion, I identify and define my key constructs in Table 1 and illustrate my research model – a single-level model operating at the group level (Kozlowski and Klein, 2000) – in Figure 1.

Construct	Definition
Time underestimation	The original definition of the planning fallacy refers to people's tendency to underestimate the *time* required to complete a project despite having historical evidence to the contrary (Kahneman and Tversky, 1979). Since then, the definition of the planning fallacy has expanded to include people's tendency to underestimate the *costs* and *risks* of future actions and to overestimate their *benefits* (Kahneman, 2011; Lovallo and Kahneman, 2003). In the remainder of this study, I retain the original definition, *time underestimation*, when I refer to the planning fallacy.
Scope overload	*Scope overload* refers to the excessive inclusion of software features within the scope of a project while consuming more resources than are available (Bjarnason, Wnuk and Regnell, 2012; Shmueli et al. 2016).
Code debt	While technical debt was initially concerned with software implementation (Li et al., 2015), it has gradually extended to include software architecture, design, documentation, requirements, and testing (Brown et al., 2010). In the remainder of this study, I focus on *code debt* – poorly written code that violates coding best practices or coding rules (Li et al., 2015) – when I refer to technical debt.

Table 1. Definition of Constructs

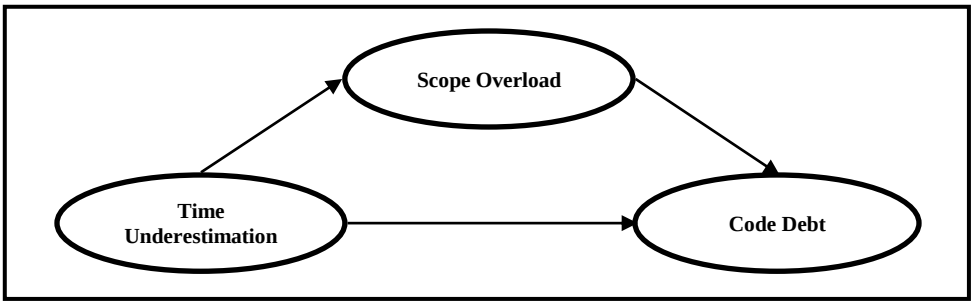

Figure 1. Research Model

Hypotheses

First, I argue that there is a significant *total effect* of time underestimation on code debt in ASD projects. To complete information technology projects successfully, project teams must skillfully manage the triple constraints of scope, time, and cost to produce a quality product, service, or result (Schwalbe, 2018; Project Management Institute, 2017). In ASD projects, the time for each iteration is typically fixed and many times so is the budget. Project teams are therefore left to consider tradeoffs between scope and quality. I argue that since agile project teams need to deliver a potentially-shippable software increment during each iteration, they will do so at the expense of quality (i.e., they will accumulate code debt). This would be exacerbated by the planning fallacy which would cause these project teams to underestimate the time required to deliver a quality product, service, or result and consequently make even further trade-offs in software quality to meet the fixed deadline. My argument is supported by Holford (2013, p. 1), who states that "if we start a development cycle with an unrealistic schedule estimate, then we can just expect to be late from the start. In an atmosphere like this, ill-considered compromises are almost inevitable. These compromises constitute the primary source of technical debt introduced to a new code base." I, therefore, hypothesize that:

H1: An increase in time underestimation will lead to an increase in code debt in ASD projects.

Second, I argue that there is a causal relationship between time underestimation and scope overload. During each iteration, ASD teams are responsible for committing to complete a set of user stories (i.e., some amount of work). While the business representative, or product owner, is responsible for creating a prioritized list of user stories to be completed, the ASD team has the latitude to determine how many user stories (in priority order) to complete during each iteration. I argue that if these teams underestimate the time required to complete the user stories, then they will select more user stories than can be reasonably delivered at the desired level of quality holding time and resources constant. This causal relationship was also proposed – but never tested – by Shmueli et al. (2016, p. 411) who argued that "if software engineers perceive each feature as less demanding in terms of the time needed for its development, they may have the impression that more features could be developed within a given project duration." I, therefore, hypothesize that:

H2: An increase in time underestimation will lead to an increase in scope overload in ASD projects.

Third, I argue that the relationship between time underestimation and code debt is mediated by scope overload. If time underestimation does, indeed, lead to scope overload as I have argued in Hypothesis 2, then ASD teams will find themselves with more scope to deliver than there is time to deliver it at the at the desired level of quality holding time and resources constant. These teams may, therefore, take shortcuts, such as omitting to comment code and ignoring coding best practices, which will lead to the accumulation of code debt. I, therefore, hypothesize that:

H3: The effect of time underestimation on code debt in ASD projects is mediated by scope overload.

RESEARCH METHODOLOGY

I will test my hypotheses using a randomized lab experiment with approximately 150 ASD practitioners (50 in each treatment/control group) recruited through Qualtrics. This method is appropriate for my study because experiments are especially powerful for making inferences about causal relationships due to their high internal validity (Shadish, Cook and Campbell, 2002; Trochim and Donnelly, 2006).

Experimental Design

I will use a *longitudinal design* (Shadish et al., 2002), which will allow me to examine the effects of the planning fallacy during each of the three sprints (or time-periods) separately, to test whether participants in the control group improve their estimation from one sprint to the next (i.e., a learning effect), and to test whether the effect of the manipulation persists from T2 to T3 or if it dissipates once attention is no longer drawn to the distributional information (i.e., outside view). This design is illustrated in Figure 2 using the notation from Shadish et al. (2002), in which R represents the random assignment of units, X represents a treatment (subscripts identify different treatments), and O represents either a pre-test or a post-test.

		T1		*T2*	*T3*
Treatment Group A	**R**	**O**	$\mathbf{X_A}$	**O**	**O**
Treatment Group B	**R**	**O**	$\mathbf{X_B}$	**O**	**O**
Control Group	**R**	**O**		**O**	**O**

Figure 2. Experimental Design

Experimental Procedure

My experiment involves collecting data for three sprints: T1, T2, and T3. At the beginning of the experiment, participants will be randomly assigned to one of three groups: Treatment Group A (which will receive the outside view manipulation), Treatment Group B (which will receive the inside view manipulation), and Control Group (which will receive no manipulation). At the beginning of each sprint, participants will be given the same experimental scenario and asked to identify a set of incomplete tasks that they plan to complete in the current sprint. Participants will then be asked to estimate the time that they will need to complete each task. At the end of each sprint, participants will be asked to provide the actual time that it took to complete each task. Participants in Treatment Group A will be asked to look across other projects in the past and present and focus on distributional information (including summary data from all participants' responses in the first sprint, T1) when providing estimates, participants in Treatment Group B will be asked to look forward and focus on singular information relevant to the set of tasks at hand, and participants in the Control Group will simply move from the first sprint (T1) to the second sprint (T2) without any manipulations.

Measures

My measures are described in Table 2 along with key references from which they were derived.

Construct	**Description of Measure**	**Key References**
Time underestimation	I will compute time underestimation (in hours) as the difference between the *actual time* to complete the agreed-upon project scope and the *estimated time* to complete the agreed-upon project scope for each iteration as reported by the project teams.	Shmueli et al. (2016)
Scope overload	I will measure scope overload as the *total number of features* that the project team commits to delivering for each iteration.	Shmueli et al. (2016)
Code debt	I will measure code debt as the *number of hours* required to resolve all issues that were identified in the source code for each iteration as computed by a source code analyzer such as SonarQube (SonarSource, 2017).	Letouzey (2012); Letouzey and Ilkiewicz (2012); Luhr (2015)

Table 2. Description of Measures

ANALYSIS AND EXPECTED RESULTS

I will test my mediation model using a series of regressions following the *path analytic approach* by Shrout and Bolger (2002) instead of the classical *causal steps approach* by Baron and Kenny (1986), which has met with several criticisms since it was initially proposed. Also, I will conduct manipulation checks using a combination of *t-tests, ANOVA, and ANCOVA* (Field, 2013) to determine whether time underestimation differs between the treatment and control groups during the various sprints. While I do not expect to find any significant differences at T1 before the treatment is administered, I expect to find a significant difference at T2 after the treatment has been administered. Whether I detect a significant difference between the groups at T3 will depend on whether the treatment effect persists. Further, I will conduct control checks to determine whether demographic factors such as age, gender, and work experience contribute to any significant differences between the treatment and control groups.

EXPECTED CONTRIBUTIONS

Implications for Research

My primary contribution to research will stem from answering my first research question: *What is the impact of the planning fallacy on the accumulation of technical debt in ASD projects?* Studies linking the planning fallacy and technical debt, or even time underestimation and code debt, are virtually non-existent in the literature. However, there seems to be an important link between these two research areas. Establishing a relationship between the planning fallacy and technical debt would represent an important contribution to research because it would advance our knowledge about the causes of technical debt. Also, linking the nomological networks of the planning fallacy and technical debt would open the door for researchers to explore various ways to leverage existing knowledge about the planning fallacy as potential solutions to managing technical debt in ASD projects.

Another key contribution to research would stem from answering my second research question: *How does the planning fallacy contribute to the accumulation of technical debt in ASD projects?* By identifying scope overload as an intervening mechanism between time underestimation and code debt, I would have not only established a causal relationship between time underestimation and scope overload (which has not been previously tested), but I would have also provided a glimpse into the black box connecting time underestimation and code debt. This would open the door for other researchers to identify additional mediators and possibly even moderators of the relationship between the planning fallacy and technical debt.

Implications for Practice

Technical debt is a real and significant business challenge. Therefore, studies that identify ways to manage technical debt represent important contributions to practice. By finding empirical support for my research model, I would inform practice of the role that the planning fallacy can have in contributing to the accumulation of technical debt in their ASD projects. Also, by demonstrating that the use of reference information (an outside view mechanism) can mitigate the effects of time underestimation on scope overload and code debt, I would provide practice with a technique that can be immediately applied to reduce technical debt in their ASD projects. Further, by mitigating the effects of the planning fallacy on technical debt in ASD projects, developers, customers, and end users are likely to be more satisfied with the final product, service, or result.

Limitations and Future Research

Like all studies, this one is not without limitations. First, by deciding to conduct a lab experiment, I have consciously decided to sacrifice some external validity for greater internal validity. Second, recruiting practitioners is costly, and therefore the number of participants will be constrained by funding that is available for this study. Third, the manipulation may not be robust enough to detect small but real effects in the sample. Fruitful avenues for future research include examining whether the findings from this study hold in field settings, whether several outside view mechanisms could be combined to create more potent manipulations, and whether other decision-making biases interact with the planning fallacy.

REFERENCES

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spínola, R. O., Shull, F. and Seaman, C. (2016) Identification and Management of Technical Debt: A Systematic Mapping Study, *Information and Software Technology*, 70, 100-121.

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A. and Avgeriou, P. (2015) The Financial Aspect of Managing Technical Debt: A Systematic Literature Review, *Information and Software Technology*, 64, 52-73.

Archer Software (2016a) How Much Does It Cost to Develop Smart House Software? (available at http://www.archer-soft.com/en/blog/how-much-does-it-cost-develop-smart-house-software; retrieved September 15, 2018).

Archer Software (2016b) Software Development for Self Driving Cars (available at http://www.archer-soft.com/en/blog/software-development-self-driving-cars; retrieved September 15, 2018).

Baron, R. M. and Kenny, D. A. (1986) The Moderator-Mediator Variable Distinction in Social Psychological Research: Conceptual, Strategic, and Statistical Considerations, *Journal of Personality and Social Psychology*, 51, 6, 1173-1182.

Behutiye, W. N., Rodríguez, P., Oivo, M. and Tosun, A. (2017) Analyzing the Concept of Technical Debt in the Context of Agile Software Development: A Systematic Literature Review, *Information and Software Technology*, 82, 139-158.

Bjarnason, E., Wnuk, K. and Regnell, B. (2012) Are You Biting Off More Than You Can Chew? A Case Study on Causes and Effects of Overscoping in Large-Scale Software Engineering, *Information and Software Technology*, 54, 10, 1107-1124.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I. and Sangwan, R. (2010) Managing Technical Debt in Software-Reliant Systems, *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, 47-52.

Buehler, R. and Griffin, D. (2015) The Planning Fallacy: When Plans Lead to Optimistic Forecasts, in M. D. Mumford and M. Frese (Eds.) *The Psychology of Planning in Organizations: Research and Applications*, New York, NY, USA, Routledge, 31-57.

Buehler, R., Griffin, D. and Peetz, J. (2010) The Planning Fallacy: Cognitive, Motivational, and Social Origins, in M. P. Zanna and J. M. Olson (Eds.) *Advances in Experimental Social Psychology*, 43, 1-62.

Buehler, R., Griffin, D. and Ross, M. (1994) Exploring the "Planning Fallacy": Why People Underestimate Their Task Completion Times, *Journal of Personality and Social Psychology*, 67, 3, 366-381.

Camden, C. (2013) Avoid getting buried in technical debt (available at https://www.techrepublic.com/blog/software-engineer/avoid-getting-buried-in-technical-debt/; retrieved September 15, 2018).

Cunningham, W. (1992) The Wycash Portfolio Management System, *ACM SIGPLAN OOPS Messenger*, 4, 2, 29-30.

Curtis, B., Sappidi, J. and Szynkarski, A. (2012) Estimating the Principal of an Application's Technical Debt, *IEEE Software*, 29, 6, 34-42.

Field, A. (2013) Discovering Statistics Using IBM SPSS Statistics, SAGE.

Flyvbjerg, B. (2013) Quality Control and Due Diligence in Project Management: Getting Decisions Right by Taking the Outside View, *International Journal of Project Management*, 31, 5, 760-774.

Fowler, M. (2009) Technical Debt Quadrant (available at http://martinfowler.com/bliki/TechnicalDebtQuadrant.html; retrieved September 15, 2018).

Gartner (2010) Gartner Estimates Global 'It Debt' to Be $500 Billion This Year, with Potential to Grow to $1 Trillion by 2015 (available at https://www.gartner.com/newsroom/id/1439513; retrieved September 15, 2018).

Guo, Y., Spínola, R. O. and Seaman, C. (2016) Exploring the Costs of Technical Debt Management – a Case Study, *Empirical Software Engineering*, 21, 1, 159-182.

Holford, M. (2013) Technical Debt and the Planning Fallacy (available at http://www.alexanderinteractive.com/blog/2013/02/technical-debt-planning-fallacy; retrieved August 2, 2016).

Kahneman, D. (2011) Thinking, Fast and Slow, Farrar, Straus and Giroux.

Kahneman, D. and Lovallo, D. (1993) Timid Choices and Bold Forecasts: A Cognitive Perspective on Risk Taking, *Management Science*, 39, 1, 17-31.

Kahneman, D. and Tversky, A. (1979) Intuitive Prediction: Biases and Corrective Procedures, *Management Science*, 12, 313-327.

Kahneman, D. and Tversky, A. (1982) Intuitive Prediction: Biases and Corrective Procedures, in D. Kahneman, P. Slovic and A. Tversky (Eds.) *Judgment under uncertainty: Heuristics and biases*, Cambridge, England, Cambridge University Press, 3-20.

Kozlowski, S. W. J., and Klein, K. J. (2000) A multilevel approach to theory and research in organizations: Contextual, temporal, and emergent processes, in K. J. Klein and S. W. J. Kozlowski (Eds.) *Multilevel theory, research, and methods in organizations: Foundations, extensions, and new directions*, San Francisco, CA, USA, Jossey-Bass, 3-90.

Kruchten, P., Nord, R. L., Ozkaya, I. and Falessi, D. (2013) Technical Debt: Towards a Crisper Definition Report on the 4th International Workshop on Managing Technical Debt, *ACM SIGSOFT Software Engineering Notes*, 38, 5, 51-54.

Letouzey, J. L. (2012) The SQALE Method for Evaluating Technical Debt, *Third International Workshop on Managing Technical Debt*, 31-36.

Letouzey, J. L. and Ilkiewicz, M. (2012) Managing Technical Debt with the SQALE Method, *IEEE Software*, 29, 6, 44-51.

Li, Z., Avgeriou, P. and Liang, P. (2015) A Systematic Mapping Study on Technical Debt and Its Management, *Journal of Systems and Software*, 101, 193-220.

Lim, E., Taksande, N. and Seaman, C. (2012) A Balancing Act: What Software Practitioners Have to Say About Technical Debt, *IEEE Software*, 29, 6, 22-27.

Lovallo, D. and Kahneman, D. (2003) Delusions of Success, *Harvard Business Review*, 81, 7, 56-63.

Luhr, R. L. (2015) The Application of Technical Debt Mitigation Techniques to a Multidisciplinary Software Project, Doctorate, Montana State University-Bozeman, College of Engineering.

Pisipati, U. (2017) Automatic Speech Recognition, Virtual Assistants and the Rise of Conversational Era (available at https://medium.com/@udaynag/automatic-speech-recognition-virtual-assistants-and-the-rise-of-conversational-era-1493a224d4eb; retrieved September 15, 2018).

Project Management Institute (2017) A Guide to the Project Management Body of Knowledge (PMBOK Guide) – Sixth Edition, Project Management Institute.

Schwalbe, K. (2018) Information Technology Project Management, Cengage Learning.

Shadish, W. R., Cook, T. D. and Campbell, D. T. (2002) Experimental and Quasi-Experimental Designs for Generalized Causal Inference, Wadsworth Publishing.

Shmueli, O., Pliskin, N. and Fink, L. (2016) Can the Outside-View Approach Improve Planning Decisions in Software Development Projects?, *Information Systems Journal*, 26, 4, 395-418.

Shrout, P. E. and Bolger, N. (2002) Mediation in Experimental and Nonexperimental Studies: New Procedures and Recommendations, *Psychological Methods*, 7, 4, 422-445.

SonarSource (2017) Continuous Inspection | SonarQube (available at https://www.sonarqube.org; retrieved September 15, 2018).

Trochim, W. M. K. and Donnelly, J. P. (2006) The Research Methods Knowledge Base, Atomic Dog.