The Journal of Systems and Software 135 (2018) 1–16

Contents lists available at ScienceDirect

The Journal of Systems and Software

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

Managing architectural technical debt: A unified model and systematic literature review

Terese Besker*, Antonio Martini, Jan Bosch

Computer Science and Engineering, Software Engineering, Chalmers University of Technology, Gothenburg, Sweden

ARTICLE INFO

Article history:
Received 20 December 2016
Revised 21 September 2017
Accepted 25 September 2017
Available online 28 September 2017

Keywords:
Systematic literature review
Architectural technical debt
Software maintenance
Software architecture

ABSTRACT

Large Software Companies need to support the continuous and fast delivery of customer value in both the short and long term. However, this can be impeded if the evolution and maintenance of existing systems is hampered by what has been recently termed Technical Debt (TD). Specifically, *Architectural* TD has received increased attention in the last few years due to its significant impact on system success and, left unchecked, it can cause expensive repercussions. It is therefore important to understand the underlying factors of architectural TD. With this as background, there is a need for a descriptive model to illustrate and explain different architectural TD issues. The aim of this study is to synthesize and compile research efforts with the goal of creating new knowledge with a specific interest in the architectural TD field. The contribution of this paper is the presentation of a novel descriptive model, providing a comprehensive interpretation of the architectural TD phenomenon. This model categorizes the main characteristics of architectural TD and reveals their relations. The results show that, by using this model, different stakeholders could increase the system's success rate, and lower the rate of negative consequences, by raising awareness about architectural TD.

1. Introduction

In 1992, Cunningham (1992) introduced the financial metaphor of Technical Debt (TD) to describe the need for recognizing the potential long-term and far-reaching negative effects of immature code, made during the software development lifecycle, which has to be repaid with interest in the long term. Cunningham used the financial terms of debt and interest when describing TD: "*Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.*" Another, more recent, definition was provided by Avgeriou et al. (2016b) who define TD as *"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability."*

Today, large-scale software companies strive to increase their efficiency in each lifecycle phase, by reducing time and resources deployed by the development teams. To achieve this goal of delivering high-quality systems, the software architecture is especially important and should contribute to a minimal maintenance effort. Van Vliet (2008) states that maintenance activities consume 50–70% of the total effort spent during a typical software project. Left unchecked, these maintenance activities can make the architecture diverge towards a suboptimal state, and, in the worst-case scenario, towards system obsolescence or crisis (Martini et al., 2014).

During the development of large-scale systems, software architecture plays a significant important role (Kruchten et al., 2012) and consequently a vital part of the overall TD relates to suboptimal architectural decisions and is regarded as Architectural Technical Debt (ATD) (Besker et al., 2017a; Martini and Bosch, 2015a). ATD is primarily incurred by architectural decisions with the consequence of immature architectural artifacts and compromised quality attributes (Besker et al., 2017b). ATD commonly refers to violations of best practices (Martini et al., 2014), consistency and integrity constraints of the architectures, or the implementation of immature architecture techniques. These consequences primarily result from the compromise of modularity, reusability, analyzability, modifiability, testability, or evolvability during software architecting (Li et al., 2016). ATD does not always receive the full attention from the architect and management

* Corresponding author.
E-mail addresses: besker@chalmers.se (T. Besker), antonio.martini@chalmers.se (A. Martini), jan@janbosch.com (J. Bosch).

https://doi.org/10.1016/j.jss.2017.09.025
0164-1212/

teams due to the fact that ATD typically concerns the cost of the long-term maintenance and evolution of a software system instead of the visible short-term business value. Another reason for this is that ATD is hard to identify and measure since it is not easily visible (Kruchten, 2012) within the code base. The visibility of ATD, generally first occurs when the system significantly reveals shortcomings or complications in the maintenance or operation (Li et al., 2014a).

TD management involves navigating a path that considers both value and cost, to focus on overall ROI over the software lifecycle (Nord et al., 2012) for all different types of TD, and, although a great deal of theoretical work on the *architectural* aspects of TD has recently been produced, practical ATD Management (ATDM), with an architectural focus, lacks empirical studies (Martini et al., 2016a).

ATD is evidently detrimental to software companies, and, since ATD has such a significant negative impact on software systems (Besker et al., 2017b; 2017a), it is important to understand what it is and of what it is composed. This study focuses on different ATD categories and their related effects, and thereby becomes highly relevant when providing a platform for analyzing different ATDM strategies, solutions and challenges. To date, we have not found any studies describing this issue in a unified way, which could facilitate the challenges of understanding and managing ATD in an overall context. Research can benefit from research synthesis techniques that help summarize and assess the body of results accumulating in the literature (Zimmermann, 2016). Therefore, to explore and understand these concerns in a more comprehensive context, a systematic literature review is conducted in the area of ATD, with research questions focusing on the current knowledge regarding debt, interest, principal and existing challenges, and solutions in managing ATD.

This unified model and literature review can be of benefit from a variety of academic perspectives. For researchers interested in the architectural aspects of TD, the research agenda for ATD helps to build upon already existing work and guide efforts towards new research directions. For practitioners, the unified model can help to identify ATD and to evaluate what problems might occur while dealing with ATD and the consequences if these challenges are left unattended.

The objective of this study has been achieved through employing a thorough Systematic Literature Review (SLR) research method (Kitchenham et al., 2009). SLR is a well-established research method for conducting a structured and systematic way of performing a review using a protocol by formally defining each process within the review process.

The main objective of this study is to clarify and contribute to an extended knowledge base in the research area of ATD and to create a common platform for future research. The contributions of this study are as follows:

1. We present results showing that there is no one unified and overarching description or interpretation for ATD and, therefore, a 'state-of-the-art' review of significant issues is provided, concerning various ATD issues.
2. This study identifies aspects of previous studies, and examines how the studies have been conducted.
3. This study provides a novel descriptive model that provides an overall understanding concerning knowledge currently of interest in the research area of ATD. This unified descriptive model can support the process of more informed management of the software development lifecycle, with the goal of raising the system's success rate and lowering the rate of negative consequences for both the academic and practitioner community.
4. Having this visual and unified model in which stakeholders can rapidly obtain a holistic overview is valuable. Researchers and practitioners can use this unified model to evaluate and understand what problems might occur while dealing with ATD and the consequences if these challenges are left unattended. In our unified model, we provide a checklist of the aspects of ATD reported in the literature. Researchers and practitioners can use this checklist as a general reference tool for recognizing ATD.
5. This study shows that there is a compelling need for supporting tools and methods for system monitoring and evaluating ATD, but also shows that no software tools covering the full spectrum of ATD are yet available.
6. This study provides new insights into the refactoring of ATD research by showing that practitioners, in general, lack strategies for architectural refactoring, and, therefore, such an activity might result in an ad-hoc process where the results are inadequate. In this paper, we provide the key dimensions that need to be taken into consideration when defining such a refactoring strategy.
7. To both practitioners and academics, this study demonstrates the relevance of paying more attention and effort to remediating ATD during the software lifecycle, in order to decrease the level of negative impact due to ATD on daily software development work.

This paper is structured in nine sections. The following section introduces background information that is used during a discussion of the results. In the third section, the SLR method is described. The fourth section presents the results from the retrieval of publications, and section five presents the results of the data collection of the publications. Section six addresses the importance of ATD and the need for a unified model. Section seven discusses the results of both the literature review and the unified model and analyzes the results of an ATD research agenda and lists the threats to the validity. Section eight reviews the related research, while the last section, nine, concludes the paper.

This paper is an extension of the previously published paper that was originally published at the 42nd Euromicro Conference on Software Engineering and Advanced Applications (SEAA) in 2016 (Besker et al., 2016). However, this paper includes several more surveyed publications, since the timeframe when searching for publications was expanded to also include the year 2016. We have also added both a forward and backward snowballing technique for this extra time period. This paper also includes an additional research question, and, consequently, new findings, and a more in-depth analysis of all of the research questions has been obtained. Moreover, the unified model offered is an extention of the previous model presented in the abovementioned conference paper (Besker et al., 2016), where the model has been enhanced by including additional research question (importance of ATD - RQ1) and updated research results for all of the different aspects in the model.

2. Background

In order to provide the reader with the necessary information that is needed to better understand the remainder of the paper, this section provides a background to the ATD domain. In this broad view, we examine what constitutes ATD in terms of debt, interest, and principal and how it is managed with regard to related management processes, current challenges and analyzing support.

There are different types of TD, with Alves et al. (2014) having identified and organized the different types by considering the nature as a classification criterion for each TD type. They identified 13 different types of TD, including Architectural TD, Build Debt, Infrastructure TD, Requirement TD, Test Automation TD, and Code TD. Alves et al. define ATD as referring *"to the problems encountered in project architecture, for example, violation of modularity,*