Thesis for the Degree of Licentiate of Engineering

An Empirical Investigation of the Harmfulness of Architectural Technical Debt

Terese Besker

Division of Software Engineering
Department of Computer Science and Engineering
Chalmers University of Technology and Göteborg University
Göteborg, Sweden, 2018

An Empirical Investigation of the Harmfulness of Architectural Technical Debt

Terese Besker

Technical report 172L
ISSN 1652-876X
Department of Computer Science and Engineering
Division of Software Engineering
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Göteborg
Sweden
Telephone + 46 (0)31-772 1000

Printed by Chalmers Reproservice,
Göteborg, Sweden 2018.

“Gratitude is not only the greatest of virtues, but the parent of all others”

Marcus Tullius Cicero

Abstracts

Background: In order to survive in today's fast-growing and ever fast-changing business environments, large-scale software companies need to deliver customer value continuously, both from a short- and long-term perspective. However, the consequences of potential long-term and far-reaching negative effects of shortcuts and quick fixes made during the software development lifecycle, described as Technical Debt (TD), can impede the software development process.

Objective: The overall goal of this Licentiate thesis is to *empirically* study and understand *in what way* and *to what extent*, TD in general and architectural TD specifically, influence today's software development work and, specifically, with the intention of providing more *quantitative* insights into the field.

Method: To achieve the objectives, a combination of both quantitative and qualitative research methodologies are used, including interviews, surveys, a systematic literature review, a longitudinal study, correlation analysis, and statistical tests. In five of the seven included studies, we use a combination of multiple research methods to achieve high validity.

Results: We present results showing that software suffering from TD will cause various different negative effects on both the software and on the developing process. These negative effects can be illustrated from a technical, a financial and from a developer's working situational perspective.

Conclusion: This thesis contributes to the understanding and quantification of *in what way* and *to what extent* TD is harmful to software development organizations. The results show that software practitioners estimate that they waste 36% of their working time due to experiencing TD and that the TD is causing them to perform additional time-consuming work activities. This study also shows that, compared to all types of TD, architectural TD has the greatest negative impact on the daily software development work.

Keywords

Software Engineering, Empirical Research, Technical Debt, Software Architecture, Software Quality, Software Developing Productivity, Morale, Mixed-methods

Acknowledgements

First of all, I would like to express my deepest gratitude and appreciation to my main supervisor, Professor Jan Bosch, for his encouragement, support, guidance, and engagement. You continuously raise the bar with me, and, most importantly, make me believe I can reach my goals.

Next, I would like to express my sincere appreciation to my second supervisor, Professor Antonio Martini, for always sharing his technical knowledge and expertise. Besides being a great friend, your support, ideas and comments have significantly improved the quality of my research.

Many thanks also go to Professor Helena Holström Olsson for her sincere support whenever I needed it. Without my supervisors' support, this work would never had been accomplished.

I would also like to thank my family and all my friends for their support and sacrifices to ensure that I could pursue this dream.

Finally, I would like to thank all the partners at the Software Center for supporting my research and ensuring that we conduct research into highly relevant topics from both an academic and software industrial perspective.

List of Publications

Appended Papers

This thesis is based on the following papers:

[A] T. Besker, A. Martini, and J. Bosch, “Managing architectural technical debt: A unified model and systematic literature review”, *Journal of Systems and Software, vol. 135, pp. 1-16, 2018.*

[B] T. Besker, A. Martini, and J. Bosch,T. Besker, A. Martini, and J. Bosch, “Time to Pay Up - Technical Debt from a Software Quality Perspective”, *In proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ ICSE17, 2017.*

[C] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt - When and by whom will it be paid?”, *Proceedings of IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, pp. 13-23, 2017.*

[D] T. Besker, A. Martini, and J. Bosch, "Impact of Architectural Technical Debt on Daily Software Development Work - A Survey of Software Practitioners, "*Proceedings in 43th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Vienna, 2017, pp. 278-287.*

[E] T. Besker, A. Martini, and J. Bosch, “Technical Debt Cripples Software Developer Productivity - A longitudinal study on developers’ daily software development work”, *In submission to the First International Conference on Technical Debt @ ICSE18, 2018.*

[F] H. Ghanbari, T. Besker, A. Martini, and J. Bosch, “Looking for Peace of Mind? Manage your (Technical) Debt - An Exploratory Field Study”, *Proceedings in the International Symposium on Empirical Software Engineering and Measurement (ESEM), Toronto, Canada, 2017*

[G] A. Martini, T. Besker, and J. Bosch, “Technical Debt Tracking: Current State of Practice - A Survey and Multiple Case-Study in 15 large organizations”, *Journal of Science of Computer Programming, accepted for publication 2017.*

Other Publications

The following papers are published but not appended to this thesis:

[A] T. Besker, A. Martini, and J. Bosch, "A Systematic Literature Review and a Unified Model of ATD." *Proceedings in 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), Cyprus, 2016, pp. 189-197.*

[B] T. Besker, A. Martini, J. Bosch, and M. Tichy, "An investigation of technical debt in automatic production systems," *Proceedings of the XP2017 Scientific Workshops, Cologne, Germany, 2017.*

[C] A. Martini, T. Besker, and J. Bosch, "The introduction of Technical Debt Tracking in Large Companies", *Proceedings in the 23rd Asia-Pacific Software Engineering Conference (APSEC), Hamilton, New Zealand, 2017*.

Personal Contribution

For all included publications, the first author is the main contributor with regards to the inception, planning and execution of the research, and the writing of this publication. The same applies to the excluded publications for which I am the first author. For the two publications in which I am listed as second co-author, the following contributions were made by me:

Ghanbari et al.: In this paper, I participated in the design of the overall study, I conducted two of the interviews, I designed, implemented and partly analyzed the survey, and I contributed in writing the publication.

Martini et al.: In this paper, I designed and implemented the survey, I contributed during the data analysis phase, and I contributed in writing the publication.

CONTENTS

1. Introduction

In order to survive in today's fast-growing and ever fast-changing business environments, large-scale software companies need to deliver customer value continuously, both from a short- and long-term perspective. During the software development lifecycle, companies need to consider the tradeoffs between the overall quality of the software, and the costs of the software development process in terms of the required time and resources. In general, software companies strive to balance the quality of the software with the ambition of increasing the efficiency and decreasing the costs in each lifecycle phase, by reducing time and resources deployed by the development teams.

Examples of this tradeoff can be illustrated by scenarios where software companies deliberately implement sub-optimal solutions in order to shorten the time-to-market or when resources are limited in practice, by implementing "quick fixes" or "cutting corners" during the software development process. Even if the best intention is to go back and refactor the sub-optimal solution immediately afterward, there is a tendency that these refactoring tasks will be postponed since, commonly, there are other important deadlines in the near future, where these refactoring tasks are often down-prioritized. There is also the scenario where sub-optimal solutions are implemented unintentionally, due to a lack of knowledge, guidelines or best practices.

As a result of these scenarios, the sub-optimal solutions in the software gradually grow, and the short-term implemented quick fixes in the code base live on and become more deeply embedded. Last minute hacks remain in the code and turn into features that the users depend upon, and documentation and coding conventions are perhaps also ignored, and eventually the original architecture degrades and becomes obfuscated [131]. When new requirements start appearing that necessitate the software being extended and altered, these implemented sub-optimal solutions can impede both innovation and expansion of the software system.

The result of this impediment is the accrual of what is described as Technical Debt (TD). The TD metaphor was first coined at OOPSLA '92 by Ward Cunningham [8], to describe the need to recognize the potential long-term negative effects of immature code that is made during the software development lifecycle. Cunningham used the financial terms debt and interest when describing the concept of 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.*"

An additional, a more recent, definition was provided by Avgeriou et al. [10] 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".

As an illustration of a *"technical context where the future changes are more costly or impossible"* can be exemplified by a situation where the software experiencing TD becomes fragile in terms of when unexpected side-effects occur or when changes to one part of the software cause unpredicted failures in its unrelated parts. This situation could make the software practitioners avoid altering the software, with the result of, for example, maintenance complications.

If the technical context refers to the architecture of the system, this can be illustrated by a situation where the architecture is inflexible in terms of resistance to changeability. Without first implementing extensive, costly, risky and time-consuming architectural refactoring, the possibility to implement new features is reduced significantly. In a worst-case scenario, software companies could reach a point where they have accrued so much TD that they spend more time maintaining and containing their legacy software than adding new features for their customers [131]. Accumulated negative consequences of TD can even lead to a crisis point when a huge, costly refactoring or a replacement of the entire software needs to be undertaken [103].

In conclusion, TD is considered to be detrimental to the long-term success of software development [147], and, left unchecked, TD can result in compromised quality attributes such as maintainability, reusability, performance, and the ability to add new features. In addition to potential quality complications, TD can also hinder the software development process by causing an excessive amount of wasted working time in terms of low development productivity, project delays and high defect rates [92].

However, even if the concept and harmfulness of TD are gaining importance from an academic perspective, software companies still struggle with giving TD management sufficient attention in practice. There are several major reasons for this, such as the difficulty of implementing prevention mechanics to avoid introducing TD in the first place, and to raise awareness about the negative effects TD has on the overall software development process, and difficulties in understanding and quantifying the level of negative impact from TD.

Despite the significant need for supporting tools and methods for analyzing and quantifying TD, no supporting software tools exist that iteratively include the measuring, evaluation, and tracking of different types of TD.

Consequently, the ability to quantify TD can provide a common point of reference for software practitioners when deciding upon the prioritizing of refactoring tasks and adding new features in terms of assisting the organizations in understanding the burning issues that can affect new investments and future opportunities.

Furthermore, there are several different types of TD [3],[86],[147], such as Architectural TD, Documentation TD, Requirement TD, Code TD, Test TD, and Infrastructure TD. These different TD types affect different software development parts during different development phases, and they also have different levels of negative impact on the overall software development process. This study focuses on all TD types in general, but more

specifically on the Architectural TD (ATD) type. ATD is often described as the most important source of TD [45] and also as the most frequently encountered type of TD [67].

During different phases of the overall software development process, several different professional roles are involved, including, for instance, developers, architects, testers, and product and project managers. Hence, all these roles could potentially be affected by TD in general, but each role could potentially also be more negatively affected by a specific TD type.

When studying how software practitioners are affected by TD, some studies suggest that, along with its technical and economic consequences, TD can also negatively affect developers' psychological states and morale, in terms of, for instance, confidence, optimism, enthusiasm, and loyalty [62].

The overall goal of this Licentiate thesis is to study and understand *in what way* and *to what extent,* TD and ATD (both in general, and, more specifically, from an architectural perspective), influence today's software development work from various perspectives and, specifically, with the intention of providing more quantitative insights into the field.

1.1 Background and Related Work

This thesis studies TD in general and ATD specifically, from various different perspectives, and, in order to provide the reader with the necessary information needed to better understand the remainder of the thesis, this section provides background information and describes the related work of this thesis.

Figure 1 is a conceptual model that comprehensively describes essential aspects of the concerned included research topics within the TD and the ATD domains which are used and addressed in this Licentiate thesis. As illustrated in the Figure, TD can be of different types, where some of the included papers in this thesis focus on TD in general and some of the publications have a specific focus on ATD. The Figure further illustrates that the presence of TD, causing different negative effects during the overall software development lifecycle and the presence of TD, causes the need for several different actions to be taken and knowledge to be gained for software development organizations.

This section examines the illustrated aspects in terms of different TD categorizations, different TD types, and also what constitutes ATD in terms of debt, interest, and principal, software quality attributes, software aging, different software roles, TD tracking processes, software productivity, and, lastly, the negative effects of TD in terms of developer morale are addressed.

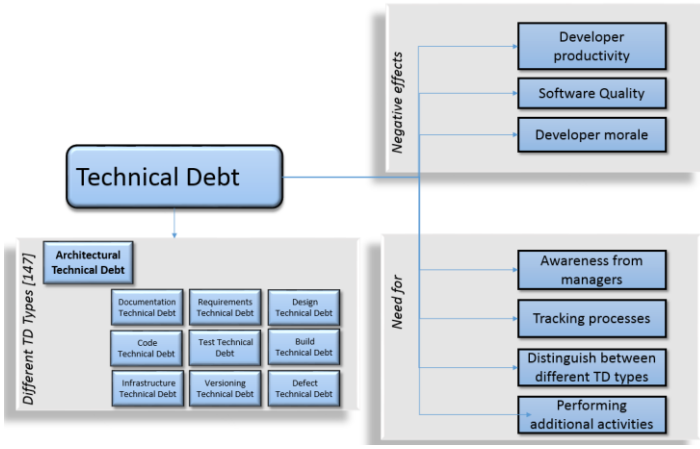

Figure 1: A conceptual model of TD and ATD, as portrayed in this thesis.

1.1.1 TD categorizations

TD can be categorized in different ways, depending on the perspective adopted. For example, Kruchten et al. [83] provide a categorization based on the visibility of different elements. As illustrated in Figure 2, their model illustrates *visible* elements such as new functionality to add and defects to fix, and the *invisible* elements (those visible only to software developers). Kruchten et al. suggest that only the invisible elements should be considered as TD, where they distinguish between evolution and quality issues.

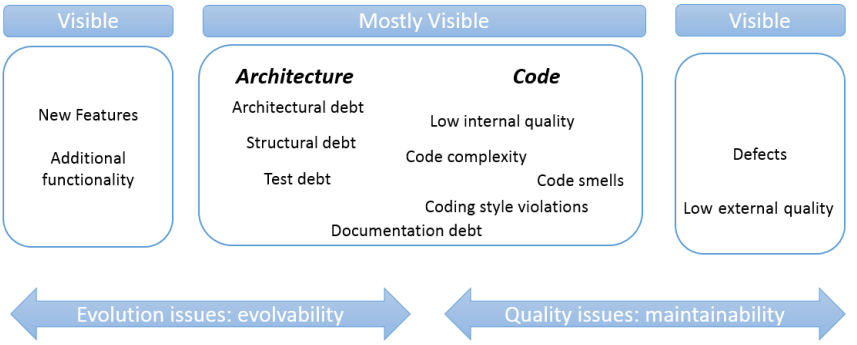

Figure 2. The TD landscape, distinguishing between evolution and quality issues [83].

Yet another classification of the TD landscape is provided by Steve McConnell [110] who categorizes TD based on whether the TD was incurred *intentionally* or *unintentionally*. The unintentionally incurred TD is the non-strategic result of doing a poor job. In some cases, this type of debt can be incurred unknowingly, for example, if a company acquires another company that has accumulated significant TD that was not identified until after the acquisition. The intentionally incurred TD is commonly found when a company makes a conscious decision to optimize for the present rather than for the future [110].

Similar to McConnell's classification, Martin Fowler [57] provides a categorization illustrated in Figure 3, where he uses a four quadrant grid considering the following characteristics: *Reckless, Prudent, Deliberate, and Inadvertent*. These characteristics comprise what is generally called the TD Quadrant and allows the classification of the debt by analyzing if it was inserted intentionally or not, and, in both cases, if it can be considered the result of a careless action or was inserted with prudence.

The *prudent* TD is deliberately introduced because the team is aware of the fact that they are taking on a TD, and puts some thought into whether the payoff for an earlier release is greater than the costs of paying it off. A team ignorant of design practices is taking on its reckless TD without even realizing the negative consequences of doing so [57].

Martin Fowler describes that *reckless* TD may not be inadvertent. A team could know about good design practices, and even be capable of practicing them, but decide to go "quick and dirty" because they think they cannot afford the time required to write clean code.

The last quadrant prudent, *inadvertent*, refers to the willingness of the teams to improve upon whatever has been done after gaining experience and relevant knowledge.

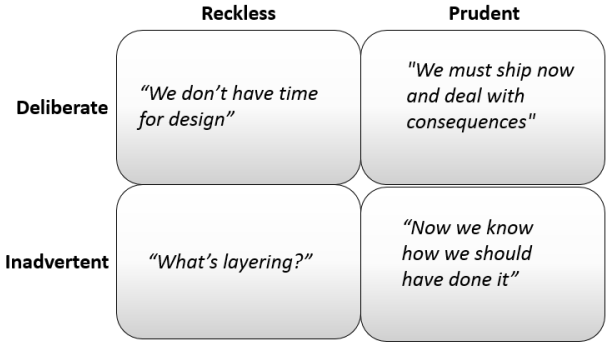

Figure 3. Technical Debt Grid Quadrant [57].

1.1.2 TD Types

There are several different types of TD, and different researchers provide different categorizations for TD types. Tom et al. [147] provide a list of seven different types of TD: code debt, design and architectural debt, environmental debt, knowledge distribution and documentation debt, and testing debt. Similar to that classification, Li et al. [86] provide an extension of, in total, 10 coarse-grained types including several sub-types of TD: Requirements TD, Architectural TD, Design TD, Code TD, Test TD, Build TD, Documentation TD, Infrastructure TD, Versioning TD, and Defect TD.

The architectural aspects of TD (ATD), which have a specific focus in this thesis, are commonly described as design decisions that, intentionally or unintentionally, compromise system-wide quality attributes, particularly maintainability and evolvability [87]. More specifically, ATD is regarded as violations of the code towards the intended architecture for supporting the business goals of the organization [98].

Alves et al. [3] define ATD as referring "*to the problems encountered in project architecture, for example, violation of modularity, which can affect architectural requirements (performance, robustness, among others). Normally this type of debt cannot be paid with simple interventions in the code, implying in more extensive development activities*."

In a similar manner, Fernández-Sánchez et al. [48] describe ATD as being caused by shortcuts and shortcomings in design and architecture or by the result of sub-optimal upfront architecture design solutions, that become sub-optimal as technologies and patterns become superseded.

However, ATD is fraught with several challenges arising from difficulties in detection [37] and the issue that ATD seldom yields observable behaviors to end users [90], and, even if there are some software tools available for analyzing TD, most of them focus on the code level instead of the architectural aspects of TD [123]. The issue of removing ATD after it has been introduced is often associated with high costs since architectural decisions take many years to evolve and are commonly made early in the software lifecycle, and it is often invisible until late in the process [82].

Furthermore, ATD tends to become widespread within the system due to what is known as vicious circles, inferring a non-linear accumulation of the interest with the result of making a later removal even more costly [98].

From a non-technical perspective, ATD is also associated with several challenges, since the awareness from both managers and other professionals about the magnitude of the related consequences of ATD are somewhat limited. This lack of knowledge often leads to the issue that ATD seldom receives sufficient attention from managers and that the allocation of both time and resources to manage and remediate ATD are limited.

1.1.3 Concepts of Debt, Principal, and Interest

The term TD is a financial metaphor, and the most common financial terms that are used in TD research are *debt*, *principal* and *interest* [5].

In financial terms, a *debt* refers to the amount of money owed by one party (debtor or borrower) to another party (creditor or lender) [6] where the obligation of the debtor is to repay a larger sum of money to the creditor at the end of that period [115]. The term debt is used to describe the gap between the existing state of a software and some hypothesized "ideal" state in which the system is optimally successful [25].

From an architectural perspective (architectural debt), this debt refers, for instance, to system shortcomings that can be improved to form an enhanced architectural software quality and to avoid excessive interest payments in the form of decreasing maintainability.

The *interest* refers to the negative effects of the extra effort that have to be paid due to the accumulated amount of debt in the system, such as executing manual processes that could potentially be automated, excessive effort spent on modifying unnecessarily complex code, performance problems due to lower resource usage by inefficient code, and similar costs [147], [37]. Ampatzoglou et al. [6] define interest in their TD financial glossary list

as: "*The additional effort that is needed to be spent on maintaining the software, because of its decayed design-time quality*."

Financially, the term *principal* refers to the original amount of money borrowed, and, from a software development perspective, the same term is used to describe the cost of remediating planned software system violations concerning TD, in other words, the cost of refactoring Ampatzoglou et al. [6]. The principal is computed as a combination of the number of violations, the hours to refactor each violation, and the cost of labor [34].

1.1.4 Software Quality Attributes

Software suffering from TD negatively affects several different quality attributes, and these affected quality attributes can, consequently, affect the software in different ways, and the level of impact can also vary during the software lifecycle [39],[161].

As depicted in Table 1, the software product quality model proposed in ISO/IEC 25010 [70] categorizes product quality properties into eight main characteristics, and each character is composed of a set of related sub-characteristics. This quality model is used in this Licentiate thesis when accessing how TD negatively affects the overall quality when experiencing TD. Li et al.'s [86] systematic mapping study shows that most examined studies argue that TD negatively affects the maintainability and that other quality attributes are only mentioned in a handful of studies.

TABLE I. SOFTWARE QUALITY ATTRIBUTES - ISO/IEC 25010

Functional suitability Completeness/Correctness/Appropriateness	**Reliability** Maturity/Availability/Fault tolerance/Recoverability
Performance efficiency Time behavior/Resource Utilization/Capacity	**Security** Confidentiality/Integrity/ Non-repudiation/Accountability/ Authenticity
Usability Appropriateness/Recognizability/ Learnability/Operability/User error protection/User interface aesthetics/Accessibility	**Maintainability** Modularity/Reusability/ Analyzability/Modifiability/ Testability
Compatibility Co-existence/Interoperability	**Portability** Adaptability/Installability/ Replaceability

1.1.5 Age of Software

Software systems are, by definition, highly evolving products, and there is a commonly held belief that the negative effects of a complex architectural design, in terms of ATD, increase with the age of the software, which is related to the concept of *software aging* [124]. Parnas [124] argues that software aging is inevitable, yet can be controlled or even reversed. Parnas highlights the causes of software aging, such as obsolescence, incompetent maintenance engineering work and the effects of residual bugs in long-running systems [40]. "*Programs, like people, get old. We can't prevent aging, but we can understand its causes, take steps to limit its effects, temporarily reverse some of the*

damage it has caused, and prepare for the day when the software is no longer viable." Furthermore, Mens et al. [112] describe that the negative effects of software aging have a significant economic and social impact in all sectors of industry and therefore it is crucial to develop tools and techniques to reverse or avoid the intrinsic problems of software aging. This notion is echoed by Lindgren et al. [93], stating *"Technical debt refers to software aging costs that are not attended to, which hence need to be repaid at a later time."*

1.1.6 Software Professional Roles

Today, there are several different kinds of professional roles present in the software industry. These roles have different working tasks and responsibilities and work in different areas and in different development phases. The different roles also can have different education, understandings and scope of knowledge. Taken together, during the software lifecycle, several different professional roles participate, and could subsequently be affected differently by TD. In this thesis, we have included the software professional roles that are affected by TD and the roles that are empowered to make a decision in the context of TD. The assessed roles are many developers, testers, architects, product managers, and product managers.

1.1.7 Tracking Process

Software tooling is a necessary component of any TD management strategy [45], and the tracking process of TD is crucial for the ability to manage TD in a proactive way. Even if there are some tools available (e.g., SonarQube), these tools usually focus on only *identifying* TD at a code level, and these code-focusing tools generally cannot prove indicative for, for example, architectural trade-off, since they can cause misleading results [96]. The available tools also rarely provide the user with any supporting information about the principal or the interest of the TD. Despite the significant need for supporting tools and methods for analyzing TD and ATD, there are no supporting software tools that exist that iteratively include the measuring, evaluation, and tracking of different types of TD. The process of starting to track TD requirements includes both costs in terms of initial investments, educational and preparation activities. However, there are some companies that have, to some extent, introduced a TD tracking process within their software development process. In this regard, Ernst et al. [45], found that only 16% of the respondents in their study used a tool to identify TD.

1.1.8 Software Development Productivity

Several publications, such as [2], [147], [86], state that TD can, in general, have a negative effect on the overall software development productivity, but these publications rarely define what productivity refers to and in what way this reduced productivity can be measured. Commonly, the existing literature relating to TD and productivity states that TD becomes a constant drain on software productivity [42], which can lead to slowing down the overall development and negatively affect productivity [147], [2]

Software systems suffering from TD are causing an extensive amount of *wasted* working time, since practitioners are forced to perform additional activities which would not be necessary if the TD was not present. In general, there are different ways of measuring software development productivity [108], and, in this Licentiate thesis, we refer to productivity as "*the ability to deliver high-quality customer value in the shortest amount of time*". This means that the less time that is wasted due to experiencing TD, the greater the increase in productivity, inferring that the practitioners can thus use more time focusing on delivering customer value.

1.1.9 Impact of TD on Developers' morale

In addition to technical and financial consequences, TD can also affect developers' morale [147], [50]. The reason for this is primarily because the occurrence of TD could hamper the developers from performing their tasks and achieving their developer goals. The term *morale* can be found within the research field of organizational sciences, management, education, and healthcare [62]. Despite the vast body of related literature, the term morale lacks a coherent and precise definition, and Hardy [62] describes that several concepts, such as satisfaction, motivation, and happiness are commonly used interchangeably to highlight the term morale. In this thesis, we have used the definition of morale, provided by Hardy [62]: *"a cognitive, emotional, and motivational stance toward the goals and tasks of a group. It subsumes confidence, optimism, enthusiasm, and loyalty as well as a sense of common purpose".* Furthermore, we adopt an approach for predicting the levels of morale from measuring a set of factors that influence morale suggested by Hardy [62], where the antecedent factors of morale are divided into three main categories: affective antecedents, future/goal antecedents, and interpersonal antecedents. Even if there are some publications mentioning the relationship between TD and developers' morale or emotions, these publications do not have this scope as their primary research focus, and none of them investigate the relationship of TD and morale using an empirical research approach.

1.2 Research Motivation

As highlighted earlier in the Introduction section, software systems and software development processes suffering from TD in general, and ATD specifically, can be impeded, in terms of the technical, financial and developer working situational perspectives. However, since limited knowledge and few supporting tools are available to measure the extent of TD within a system, it is quite difficult to compute the negative effects that TD causes in terms of, for example, extra costs, extra activities, and the need for extra resources. Without this knowledge, software development organizations are not aware of the interest that they are paying on the debt, and therefore they might not currently give TD management the necessary attention within their organizations. Furthermore, without this information, software organizations risk not focusing sufficiently on deliberate remediation of their TD, which, over time, can result in high defect rates, project delays, quality complications and very low developer productivity.

Although significant theoretical work has been undertaken to describe the negative effects of TD and ATD, to date, very few empirical studies focus on their impact and their negative effects on software development. Therefore, there is a need for more empirical assessments in the research field, with a focus on quantifying the negative effects and a more in-depth understanding of its related negative consequences. The overall goal of this Licentiate thesis is, therefore, to *empirically* study and understand *in what way* and *to what extent*, TD in general and ATD specifically, influence today's software development work and specifically with the intention of providing more *quantitative* insights into the field.

1.3 Research Goals and Research Questions

The goal of this thesis is to empirically examine the negative effects due to TD and ATD from several different perspectives, using a combination of both quantitative and qualitative research methodologies. Derived from this main goal, below are listed the four main goals, with four sub-goals formulated, which will be addressed in this thesis.

Since this thesis focuses on TD in general and on ATD more specifically, some of the research questions focus on TD (including ATD among other TD types) in general, while other research questions focus specifically on ATD.

RQ1: What is ATD and what is known in the literature about ATD?

RQ2: What is the negative impact on Software Quality due to TD?

RQ3: How do TD and ATD negatively affect practitioners during the software development process?

- **RQ3.1: How much do software practitioners estimate the negative impact on daily software development work due to TD to be?**
- **RQ3.2: How much do software practitioners estimate the negative impact on daily software development work due to ATD to be?**
- **RQ3.3: What is the negative impact on software development productivity due to TD?**
- **RQ3.4: How does TD influence developers' morale?**

RQ4: How do companies start tracking TD and what are the initial benefits and challenges?

The first research question (RQ1) set out to understand what ATD is what is known in the literature about ATD? This question will analyze how ATD is described in the body of existing research on ATD.

The second research question (RQ2) aims to address how different software quality attributes are negatively affected in software experiencing TD, and also to assess if there is a relationship between the interest of TD and the frequencies of encountering these compromised quality attributes. The investigated quality attributes are presented in Table 1.

The third research question (RQ3) seeks to address how TD and ATD negatively affect practitioners during the software development process, both by investigating how practitioners *estimate* and *perceive* the negative effects of TD and also examines how practitioners *report* similar negative effects. During this investigation, a comparison between different types of TD and different ages of the software was made, and different professional roles are also assessed. Finally, this research question examines how TD influences developers' morale during the software development process.

The fourth and final research question (RQ4), focuses on how companies start tracking TD and the initial benefits and challenges of the tracking process.

1.4 Methodology

Software engineering is a multi-disciplinary field, encompassing not only technological, but also social, boundaries. Therefore, not only do the tools and processes software engineers use need to be investigated, but also the social and cognitive processes surrounding them, which includes the study of concerned professionals, their working tasks, and activities. Thus, we need to understand how individual software engineers develop software, as well as how teams and organizations coordinate their efforts [41].

This thesis includes seven publications, and, in order to fulfill the goals of the thesis, different research methods and different research categories have been adopted. Figure 4 provides an overview of the goals with the corresponding research questions, the selected research types, the research approaches, and, finally, the research methods used for each of the included publications. It is apparent from this Figure that this thesis has a strong emphasis on empirical research, where most of the analyzed data are based on estimated and/or reported artifacts and derive knowledge from actual industrial settings and experiences rather than from theories or anecdotal evidence. It can also be seen in Figure 4 that a strong focus is placed on combining both a qualitative and quantitative research methodology using a mixed-methods approach.

Goal / Focus area	RQ	Research Type	Research Approach	Research Methods	Paper
ATD Composition *	RQ1		Qualitative (and Quantitative)	Systematic Literature Review (SLR)	A
Affected Quality Attributes due to TD	RQ2	Empirical approach	Mixed Methods	Interviews (n = 43+32), and Survey (n = 258)	B
Quantification of the interest of TD	RQ3.1	Empirical approach	Mixed Methods	Interviews (n = 32), and Survey (n = 258)	C
Software practitioners' perception of the impact of ATD*	RQ3.2	Empirical approach	Quantitative	Survey (n = 258)	D
Developers' reporting of the negative consequences due to TD	RQ3.3	Empirical approach	Mixed Methods	Longitudinal survey study (n = 43, dp = 473), Pre-study, Interviews (n = 16)	E
TD impact on Developer Morale	RQ3.4	Empirical approach	Mixed Methods	Interviews (n = 8), and Survey (n = 33)	F
Introduction of tracking TD	RQ4	Empirical approach	Mixed Methods	Interviews (n = 13), and Survey (n = 226) Document analysis	G

- This study has a specific focus on ATD

Figure 4. Overview of the included publications with corresponding goals, approaches, categorizations and methods employed .

1.4.1 Research Approaches

The included studies that form this thesis use different research approaches. The approaches adopted are listed in this section, together with a short description and benefits of each approach.

1.4.1.1 Qualitative research

The goal of conducting qualitative research is the "*Development of concepts which help us to understand social phenomena in natural (rather than experimental) settings, given due emphasis to the meanings, experiences, and views of the participants*" [129]. Our motivation for using this qualitative research approach was to obtain richer information, to gain more in-depth insights into the phenomenon we studied, and to understand the perceptions that underlie and influence different studied negative effects. The main methods for collecting qualitative data are individual interviews, group interviews, observations, and documents. In this thesis, we have chosen individual interviews, group interviews, and documents as the data collection approaches when conducting the qualitative research.

1.4.1.2 Quantitive research

The goal of conducting quantitative research is to "explain behavior in terms of specific causes (independent variables) and the measurement of the effects of those causes (dependent variables)" [63]. The benefits of a quantitative research approach include improving the generalizations of a larger number of subjects and to thereby achieve a higher objectivity. The quantitative data collection method used in this thesis is surveys.

1.4.1.3 Mixed-Methods research

A mixed-methods research approach involves the collection of both qualitative and quantitative data, where the two forms of data collection are integrated into the design through merging the data, connecting the data, or embedding the data. The purpose of this approach is to provide a more complete understanding of the phenomena being studied [113] and the benefits of a mixed-method approach can be argued to provide a stronger understanding of the problem than either by itself and by minimizing the limitations of both approaches [32]. An advantageous characteristic of conducting mixed-methods research is the ability to perform triangulation.

However, there is a potential weakness of mixing methods for the purpose of validity convergence, namely to compare outcomes from different methods to see if they agree because the interpretation of agreement or disagreement is not straightforward [113].

The mixed-methods research approach used in this thesis has contributed to a comparison of different perspectives drawn from both qualitative and quantitative data within the same studies. This approach has also provided assistance in explaining quantitative results with qualitative follow-up data collections. Even if it is claimed that it is more difficult to execute studies based on a mixed-methods approach [159], the motivation for using this

approach was to be able to address more complex research questions and to collect a richer and stronger array of evidence that could be accomplished by using a single method alone [159].

When interpreting the results from a mixed-methods research approach, there are different designs to facilitate in providing a stronger interpretation and more insight from the results. This thesis has used different typologies for the classification of different mixed-methods strategies. The *convergent parallel mixed-methods* design was used in Paper C, where we collected both qualitative and quantitative data, analyzed them separately, and compared the results to understand if the findings confirmed or contradicted each other. In Papers E and G, an *explanatory sequential mixed-methods design* was used, where we, as a first step, collected and analyzed the quantitative data and used this result to build the qualitative data collection upon. In Papers B and F, we first collected and analyzed the qualitative data and, thereafter, collected the quantitative data, using a so-called *explanatory sequential mixed method design*.

1.4.1.4 Deductive, Inductive and Abductive Reasoning

A research approach also refers to whether the research is using a deductive, inductive or an abductive reasoning approach, where the relevance of hypotheses to the study is the main distinctive point between the different approaches.

The deductive approach refers to a research approach with the objective of testing a theory rather than developing it, where the researcher advances a theory, collects data to test it, and confirms or rejects the theory [32]. This deductive research approach has been used in this thesis, when, for example, testing whether or not commonly held beliefs about software aging can be confirmed (see Paper D).

The inductive approach aims to generate meanings from the collected data in order to identify patterns and relationships to build a theory [32]. In this thesis, we have used this approach in several included publications (e.g., Papers B, C, E, and F) where we gathered detailed information from participants and then formed this information into different categories or themes. Using this inductive approach, no theories or hypotheses were applied at the beginning of these research studies, and we, as researchers, were free in terms of altering the direction for the study after the research process had begun.

Both the inductive and the deductive approaches are associated with weaknesses, for instance, in terms of a lack of clarity when selecting the theory to be tested via formulating hypotheses (deductive reasoning) or in terms of the concern that "*no amount of empirical data will necessarily enable theory-building*" [135] (inductive reasoning).

The abductive reasoning set out to address the weaknesses associated with deductive and inductive approaches to overcoming this by adopting a more pragmatic perspective. In an abductive approach, the research process starts with "surprising facts" or "puzzles", and the research process is devoted to their explanation. The researcher seeks to select the most appropriate explanation among many alternatives in order to explain these surprising facts or puzzles [26]. However, in this thesis, we have not used this abductive research approach.

1.4.1.5 Longitudinal studies

A longitudinal study is a research method that contains repetitive observations of the same variables (e.g., time usage) on more than one occasion and over time [127]. The incentive for using a longitudinal research method in this study has two principal aspects: a) To increase the precision of reporting experienced data (in our case, not based on single estimations and single perceptions). This was achieved by studying each respondent during several weeks where the reported data could be compared. Such designs are called repeated measures designs [127], and b) To examine the respondents' changing responses over time: Longitudinal designs have a natural appeal for the study of changes associated with development or changes over time. They have value for describing both temporal changes and their dependence on individual characteristics [127]. Ployhart and Vandenberg [127] state that: "*Longitudinal designs give greater precision per observation, but observations may be more expensive or difficult to collect. Problems with missing or suspect data may be harder to solve in longitudinal studies. Implementation issues also influence design, since it is not always possible to sustain the commitment of investigators and participants or the quality of study procedures".*

To address the potential problem with missing data from the respondents, for instance, if the respondents for some reason did not enter the data in one or more surveys, the respondents were always asked to report their experienced data since the last time they took the survey. This wording means that if, for some reason, the respondent did not enter the data in one or more surveys, they would enter the data from the last time the respondent took the survey. In this way, the surveys cover the full period of sampling. To sustain the commitment of the respondents, prior to starting the study, all respondents had agreed with both their managers and ourselves of their participation. All respondents who agreed to participate were sent educational material before starting the study, with the intention of minimizing inter-observer (all researchers communicate the same knowledge) and inter-instrument (all participants receive the same information) variability [127].

1.4.2 Data Collection

The collected data in this thesis consist of both primary and secondary studies, where the primary form of data collection is one which is collected for the first time by the researcher, and where the secondary study sets out to aggregate and synthesize the outcomes of other primary studies in an objective and unbiased manner using either a qualitative or quantitative form of synthesis. The secondary study in this thesis refers to the conducted the systematic literature review.

1.4.2.1 Interviews

The data collection method in this thesis includes several interviews with industrial practitioners within the software engineering field, where we, as researchers, asked a series of questions to a set of subjects about the areas of interest in the study. This thesis includes both interviews with a single interviewee, but have we also conducted several group interviews (focus group), with several interview objects at the same time. According to

the guidance provided by Runeson and Höst [133], the dialog between the researcher and the subject(s) during all interviews was conducted by a set of pre-defined interview questions.

Runeson and Höst [133], distinguish between unstructured, semi-structured, and fully structured interviews. Unstructured interviews are a very flexible approach whereby the area of interest is established by the interviewer, but the discussion of the issues is guided by the interviewees [19]. In fully structured interviews, the interviewer has full control of the order of the questions, which are all predetermined [19]. A fully structured interview is similar to a face-to-face completion of a survey [133]. The interviews conducted in this thesis are all semi-structured in nature, with the advantage of allowing for the improvisation and exploration of the studied objects [133].

Semi-structured interviews include a combination of open-ended and closed questions, designed to elicit not only the information foreseen, but also other information not foreseen by the interviewer. In semi-structured interviews, questions are planned, but are not necessarily asked in the same order as they are listed in the interview protocol [133]. We used semi-structured interviews with the intention of ensuring that they provide us with valuable results, since the interviewees' awareness and knowledge about the concept of TD could potentially differ considerably, and therefore it was important to carefully explain the concepts used in order to create a comparable understanding between the interviewer and the interviewees.

In order to obtain a more accurate rendition of the interviews, all interviews were digitally recorded and transcribed verbatim (all interviewees were asked for recording permission before starting). All interviews were treated anonymously, regarding both the name of the interviewee and the company name.

All interviews conducted were selected based on a selective sampling of the interviewees, with respect to their role and their expertise. Several of the publications included in this thesis include interviews with software roles, such as software architects, developers, testers, project managers, and product managers.

Some of the interviews conducted were characterized as "*Follow-Up*" interviews, meaning that, to some extent, they had a focus on corroborating certain findings that we already thought had been established during previous data collection activities, where the questions were carefully worded (avoiding leading questions) to allow the interviewee to provide fresh commentary to, for example, previously presented material [159].

As shown in Figure 4, the study in Paper E includes a pre-study. During this initial pre-study, the motivation for the study was presented and discussed with software practitioners from seven software companies within our network, with an extensive range of software development. This phase acted as a guide in collecting information concerning the studied context and to select the most appropriate research model to use.

The interviews in Papers C and E were conducted with interviewees who had previously answered one or more surveys, and, during these interviews, the compiled results from the interviewees' individual results from the survey were presented. During interviews with their managers and during group interviews, an aggregated view of all the respondents from the respective company was presented. This presentation allowed the interviewees

to relate to the interview questions more easily, where the results of the survey were addressed. The interview questions for these studies were designed to: a) increase the understanding of the survey results, b) ensure that the questions in the survey were understood and interpreted as intended and in a uniform manner, c) confirm the results from the survey, and d) understand the implications of the survey results.

1.4.2.2 Surveys

Initially, we would like to clarify the term "survey" in this thesis. A survey, in this context, refers to the questionnaire (to differentiate it from a "survey" as a literature review). Surveys are considered as one of the most common data collection methods in software engineering research. Surveys aim to achieve generalizability over a certain population, for instance, different software developing practitioners or end users [141]; their advantages can be described in terms of facilitating the recruitment of respondents where they can be anonymous, since the anonymity is believed to help in gaining access to normally hard to reach respondents, and it may facilitate the sharing of their experiences and opinions. Online surveys are considered useful when the issues being researched are particularly sensitive [149].

The motivation for using surveys in this thesis was to reach a high level of generalizability to a large number of software professionals, and to maximize coverage and participation without having to conduct time-consuming interviews. We also aimed to collect data for quantitative analysis that could contribute to a more detailed examination of the different relationships and aspects of the studied topics. Aside from Paper A, all papers included in this thesis incorporated a research method that, to some extent, included a survey. According to the guidance provided by Czaja and Blair [38], the drafts of all surveys were first tested by at least one industrial practitioner and by one Ph.D. candidate in order to evaluate the understanding of the questions and the usage of common terms and expressions. During this evaluation, we also monitored the time needed to complete each survey. All surveys were designed and hosted by the online survey service SurveyMonkey.

Except for the surveys used in the longitudinal study in Paper E, all the surveys used a mix of open-ended and closed questions where the respondents could either select an answer from among pre-defined alternatives and also where the respondents could formulate their answers freely in a text field. The questions were a combination of an optional and mandatory nature. To avoid bias in these surveys, the questions were developed as neutrally as possible, ordered in such a way that one question did not influence the response to the next question, and a clarifying description was provided when needed [79]. The survey invitations were mailed directly to seven companies within our networks, all located in Scandinavia, with an extensive range of software development, and invitations were also published on software engineering related networks on LinkedIn. After two weeks, a reminder was sent out to those who had been specifically invited. The surveys were anonymous, and participation in the surveys was voluntary. Due to high completion rates (~83%), we decided to reject the incomplete responses, according to the guidelines proposed by Kitchenham and Pfleeger [79].

In Paper E, we used three different surveys when conducting the study. The first survey was a start-up survey gathering descriptive statistics to summarize the backgrounds of the respondents and their companies. The second survey (longitudinal) collected repeated measures by mailing a link to an identical survey, twice a week, for 7 weeks (i.e., 14 survey occasions), and, for those respondents who did not answer within one day, a reminder was emailed. This part of the study allowed us to collect repeated measures over time, where each respondent's data were reported more than once in order to study variability over time. The third survey was a retrospective follow-up survey, in order to collect retrospective specific data from each respondent.

1.4.2.3 Systematic Literature Review

A vital part of conducting software engineering research is the ability to identify existing research on technologies, tools, theories, and methods in order to evaluate and make informed and scientific decisions. Empirical approaches that include a systematic review methodologies such as a systematic literature review (SLR), are found to be effective in this context [11]. The main rationale for undertaking an SLR is to synthesize existing work and to identify gaps in current research in order to suggest areas for further research [77].

The SLR process should be carried out in accordance with a predefined search strategy, which allows the search to be assessed. The major advantage of using this method is that the result is provided by evidence, which is robust and transferable and that sources of variation can be further studied [77]. It provides a framework for establishing the importance of the study as well as a benchmark for comparing the results with other findings [32].

1.4.3 Data analysis

The data analysis in this thesis has been carried out in different ways, depending on the type of data collection. The quantitative data are analyzed using statistics, while the qualitative data are analyzed using categorizations and sortings [133].

1.4.3.1 Quantitative Data Analysis

The techniques for analyzing and summarizing quantitative data include different methods, such as determining measures of central tendency (e.g., median and mean) and measures of dispersion (e.g., ranges and standard deviations) [52].

For example, the collected data from the surveys used in this thesis were analyzed in a quantitative fashion, i.e., by interpreting the numbers obtained from the answers. All analyses were carried out using the software package SPSS (version 22), R (version 3.3.2) [30], and by using the optional collection of R packages from tidyverse (version 1.1.1) [32] for data manipulation and visualization. The data collected in the surveys were analyzed by assessing the median, mean and standard deviation and also by using statistical methods such as Pearson's R correlation coefficient, the Pearson chi-square tests, F-tests, Holm's procedure, the Wilcoxon signed rank test and ANOVA.

Pearson's R method was used for correlation analysis of associations between variables. This method computes pairwise, determining the strength and direction of the association between two values, and can be used to describe a linear relationship between two values. Pearson chi-square tests were used for evaluating the likelihood of any observed difference between the values arising by chance, and to assess whether unpaired observations on two variables were independent of each other. F-tests using Satterthwaite's approximation of the denominator degrees of freedom were used for significance tests of regression coefficients [136]. Holm's statistical procedure [65] was used to counteract the problem of multiple comparisons of different groups of data. The Wilcoxon signed rank test is a non-parametric statistical hypothesis test and was used when comparing two related samples when the population could not be assumed to be normally distributed. The one-way ANOVA test was used to compare the means of two or more independent groups in order to determine whether there was statistical evidence that the associated means were significantly different.

1.4.3.2 Qualitative Data Analysis

When analyzing the qualitative data collected in this thesis, a thematic analysis approach was used. Thematic analysis is a method for identifying, analyzing, and reporting patterns and themes within data, which involves searching across a dataset to find repeated patterns of meaning. The thematic analysis provides a flexible and useful research tool, which offers a detailed, and yet complex, account of the collected data [22].

When analyzing the qualitative data, guidelines provided by Braun and Clarke [22] were used in order to conduct the analysis in a thorough and rigorous manner. The thematic analysis was conducted using a six-phase guide. First, the audio recorded qualitative data collected from interviews were transcribed into written form, where we were also able to familiarize ourselves with the data. The second step involved the production of initial codes from the data, where we organized the data into meaningful groups. In this phase of the analysis, a Qualitative Data Analysis (QDA) software package called Atlas.ti was used. The third phase focused on searching for themes by sorting the different codes into potential themes and collating all the relevant coded data extracts within each identified theme. Each extract of data was assigned to at least one theme and, in many cases, to multiple themes. For example, the citation "*Maybe you have to encourage the developers a bit, to get the data*" was coded as "Willingness to input data" in the theme "Measuring Wasted time Aspects" in Paper E. To ensure that the coding was performed in a consistent and reliable fashion and in order to triangulate the interpretation of the data and to avoid bias as much as possible, two authors synchronized the output of the coding, following guidelines provided by Campbell et al. [28].

The fourth phase focused on the revised set of candidate themes, involving the refinement of those themes. The refinement focused on forming coherent patterns within the themes, otherwise, we revised the themes or created a new theme. The fifth phase focused on identifying the essence of each theme and determining what aspect of the data is captured by each theme. This phase also stressed the importance of not just paraphrasing the content of the data extracts, but also identifying what is interesting about them and why. The final

phase of the thematic analysis took place when we had a set of fully developed themes, and involved the final analysis and write-up of the publication. Figure 5 graphically illustrates how the codes (light grey boxes) and the corresponding themes (darker grey boxes) were assigned in Paper E.

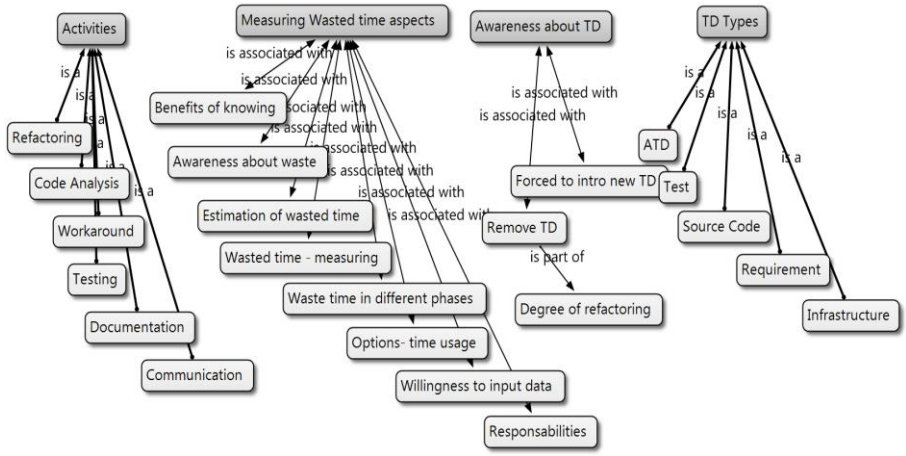

Figure 5: The thematic map from Paper E.

1.4.4 Threats to Validity

The validity of a research study refers to the trustworthiness, the credibility, the confirmability, and the data dependability of the results, and to what extent the results are reliable and not biased by the researchers' personal opinions [133], [159]. In this thesis, we have chosen a classification scheme in order to distinguish between different aspects of validity and threats to validity provided by Runesson och Höst [133], which is also used by Yin [159], and Wohlin et al. [157]. This scheme distinguishes between four aspects of validity: *construct* validity, *internal* validity, *external* validity, and *reliability.*

1.4.4.1 Construct validity

Construct validity addresses the extent to which operational measures that are studied represent what the researchers are considering and the desire to investigate according to the research questions [133], i.e., whether the theoretical constructs are interpreted and measured correctly [41]. The results presented in this thesis may be affected by some threats to construct validity, and, in order to mitigate this risk, several different approaches were employed, depending on the type of study.

For example, in the longitudinal study in Paper E, this risk was mitigated by trying to ensure that all the participants had the same base of knowledge in the field of the study, where all participants received educational material before starting the study. However, we cannot ensure that all the participants have read and understood the material. Another example is the studies which used web surveys as part of the data collection. In these studies, the construct validity threat was addressed by helping the respondents to

distinguish between different types of TD, thus a short description of each type of TD was used in the surveys. An additional threat to this validity can stem from the fact that the qualitative data derived from the survey in Papers B, C, D, F, and G, are based on perceptions and estimations (not on measured, reported or observed data) made by the respondents. Moreover, in Paper A, which is a systematic literature review, we have attempted to mitigate this risk by only including peer-reviewed publications from journals, conference proceedings, or workshop proceedings, and only two peer-reviewed book chapters were included in order to include all relevant publications concerning ATDM. In addition, in order to mitigate the risk of not retrieving relevant publications, which could affect the completeness of the study, we searched the most common electronic databases and also conducted both a forward- and backward snowballing technique.

1.4.4.2 Internal validity

The internal validity refers to whether the result is correctly derived from the researcher's conclusions without external factors potentially affecting the result. "*When the researcher is* investigating *whether one factor affects an investigated factor there is a risk that the investigated factor is also affected by a third factor"* [133]. Yin [159] states that internal validity primarily concerns studies where the researcher is trying to explain *how* and *why* one event led to another and thereby concludes a causal relationship without considering the presence of additional excluded events. Furthermore, Easterbrook et al. [41] state that it is a common mistake to confuse correlation with causality and that it is much harder to demonstrate causality than to show that two variables are correlated.

The results of this thesis could potentially be affected by this threat since, in some of the included studies, the findings are correlational and also indicate a causal relationship. For example, in the studies where we examine the amount of wasted time, activities and different TD types, this threat affects our ability to accurately explain the phenomena that we observed [159]. This threat is, for example, demonstrated in Paper B, where we use the estimated wasted time correlated with the frequency of how often the respondent encountered each of the listed quality attributes. By performing this correlation, this validity could potentially have been violated by either finding relationships that are non-existent or missing real relationships that are wrongly deemed non-significant.

1.4.4.3 External validity

The external aspect of validity addresses the extent to which it is possible to generalize the findings of the study and to be applicable to other situations. Yin [159] defines this as "*An analytic generalization consists of a carefully posed theoretical statement, theory, or theoretical proposition.*" Easterbrook et al. [41] state that, commonly, this depends on the nature of the sampling used in a study. This notion is echoed by Morse et al. [119]: "*the sample must be appropriate, consisting of participants who best represent or have knowledge of the research topic.*"

Following guidelines by Yin [159], it was important to ensure external validity was used when formulating the research questions by using the term "*how*", which increases the

possibility of arriving at an analytical generalization. Moreover, in order to mitigate this risk, the goal is to enable analytical generalization where the results are extended to situations which have common characteristics, and thus for which the findings of the study are applicable [133]. Although we cannot generalize the results in this study, we can rely on a high number of participating organizations from different types of software development settings (e.g., different business domains, programming languages, and experience) and also on a relatively high number of participants with different professional roles from each company. Furthermore, in studies where surveys are used as part of the data collection, there is always a risk that the sample is biased, and, therefore, a potential threat refers to the demographic distribution of response samples. The companies in this thesis are primarily from the Scandinavian region. Without replicating this study to other countries or regions, it is not possible to confirm the generalizability of this study.

1.4.4.4 Reliability

The goal of reliability in research is to minimize the errors and biases in a study. Reliability addresses whether the study would yield the same results if other researchers replicated them, following the same procedure, by means of the extent to which the analysis is dependent on specific researchers [133], [159]. An example of a threat to the reliability could occur if the researcher introduces bias into the study where a tool being evaluated is one in which the researchers themselves have a stake [41]. Morse et al. [119] state that it is important to ensure the attainment of rigor and reliability verification strategies early and during the whole study in a proactive way and not only apply them using a post-hoc evaluation after the research is completed.

An example of a threat to the reliability can be found in the included papers where the results are based on estimated values from participants, where we do not know what the given estimations are based on. However, as the demographic data show in these studies, many participants can count several years (more than 10) of software development experience. This means that they are used to estimate the amount of work that has been performed or that is upcoming, which mitigates the threat that the estimated effort would be very distant from the real one.

During the design of the studies included in this Licentiate thesis, we attempted to mitigate this threat in various ways. First, one prerequisite to allowing for repeatability of a study is, for example, the access to the used surveys, which we have addressed by making all surveys used available online. To assist for replicability, we have also reported how the data analysis protocol was constructed for interviews and SLR. Second, the code and sub-codes of the collected data from interviews are reported, and all surveys are made available online. Third, we have employed source triangulation, methodological triangulation, and, finally, observer triangulation (see section 1.4.2.3).

1.4.4.5 Triangulation

To achieve a higher degree of validity and reliability, we have adopted different triangulation techniques. Triangulation is important in order to increase the precision of

empirical research, in terms of taking different perspectives towards the studied object and thus providing a broader view [133]. In this thesis, three different types of triangulation are used: *source* triangulation, *observer* triangulation, and *methodological* triangulation.

Source (data) triangulation refers to using several sampling strategies to ensure that data is gathered at different times and in different situations. The use of more than one source (e.g., interviews, surveys, documents) of data makes the conclusion stronger, since it can be drawn from several sources of information [133], [114]. During source triangulation, the sources of evidence can be either of a convergence and non-convergence character, where the *converging lines of inquiry* refer to when more than one source of data corroborating the same finding and the non-convergence refers to when different sources of data address different findings [159]. As illustrated in Figure 4, this type of triangulation is used in included papers B, C, E, F, and G, where we have used more than one source of information, such as interviews, surveys, and analysis of documents. In these papers, both convergence and non-convergence source triangulation strategies are used.

Observer (researcher) triangulation refers to using more than one observer to gather and interpret data [133], [114]. This type of triangulation was achieved in all papers included in this thesis, where at least two of the involved researchers worked together with different roles during the studies, thus enabling peer debriefing and the analysis of the collected data. For example, in the conducted SLR, in Paper A, two researchers independently examined several of the retrieved publications to ensure that they were suitable and equivalently analyzed. To reduce the risk of subjectivity during the classification and extraction phase, performed by only one researcher, several publications were examined by at least two researchers in order to ensure that the returned publications were suitable and equivalently analyzed.

Methodological triangulation refers to combining different types of data collection methods, such as a combination of both qualitative and quantitative methods [133], [114]. As illustrated in Figure 4, this type of triangulation was used in this thesis included Papers A, B, C, E, F, and G, where we have used more than one type of data collection method.

1.5 Overview of Papers

In order to provide an overview of the work presented in this thesis, this section presents the included studies in this Licentiate thesis and explains how they are related.

Each paper is described in two sub-sections where the first (study summary) describes the study's goal, the research questions addressed, the selected research approach, research category, and, finally, the research method used. In a second sub-section (results) for each included paper, the results and contributions from each of the included studies are presented and synthesized. This second sub-section also sets out to discuss how the papers are related and answers the research question formulated in previous section, Section 1.3

1.5.1 Paper A: Managing Architectural Technical Debt: A unified model and systematic literature review

The following sections will briefly describe Paper A. More details of this study are reported in Chapter 2.

1.5.1.1 Study Summary

As shown in Figure 4, this research initially started by conducting an SLR. The goal of this study was to synthesize and compile the current 'state-of-the-art' in the ATD field, by conducting a systematic literature review focusing on the following research areas: ATD in terms of principal, interest, debt and related challenges and solutions for managing ATD. Brereton et al. [23] state that software engineering systematic reviews can be categorized as being qualitative in nature, and, with this information as a background, we conclude that our study is both qualitative and quantitative in nature, where the qualitative approach refers to the synthesizing process of the reviewed publication and the quantitative approach refers to the mapping of the retrieved number of publications into the study's unified model. However, the approach used in this paper has a strong emphasis on a qualitative approach and less of a quantitative one.

The study was conducted by automatically searching in six well-known digital libraries: the ACM Digital Library, IEEExplore, ScienceDirect, SpringerLink, Scopus, and Web of Science and also by a manual hand search in all the proceedings of a key conference on the subject: the International Workshop on Managing Technical Debt (MTD Workshop). Additionally, we also conducted both a forward and a backward snowballing technique to the retrieved publications.

The target of the search term was defined to search in both title and abstract, and the search term (query) contained the keywords: **"technical debt" AND architec∗**. The search was conducted in April 2017 and included publications within the timeframe of 2005-2016.

To screen out the most interesting and relevant publications for this review, a filtering technique based on five different stages was used. In the first stage, 166 publications from the different data sources were retrieved and merged. The second stage of finding and removing duplicates resulted in reducing the number of publications to 79. The third stage was applied after the full texts were retrieved, where each publication was checked using the defined inclusion and exclusion criteria. This stage returned 38 publications, for which the snowballing technique was applied in a fourth stage. In stage five, we again applied the inclusion and exclusion criteria of the publications retrieved using the snowballing technique, which returned 42 publications for a detailed quality assessment. In the sixth stage, the publications went through an assessment process, with the goal of assessing the quality of all publications. Finally, in the seventh stage, data were extracted from each of the 42 primary publications included. Based on these 42 publications, we conducted a synthesis including a descriptive synthesis and a quantitative summary (meta-analysis) by studying selected venues, numbers of publications per venue and publication types, and publications per year.

There was a wide agreement in the reviewed publications that ATD is of primary importance to software development. However, it was observed that ATD was described in a scattered and inconsistent manner. Consequently, we concluded that, in order to derive more value from the results concerning ATD and its effects, a holistic model depicting different views and their implications at hand was required. We thus constructed a novel descriptive model that provided an overall understanding of existing knowledge in the research area of ATD with the aim of providing a comprehensive interpretation of the ATD phenomenon. This model clarifies the different aspects of each research question and assembles relationships between them, together with an ATD Identification Checklist, recognized ATD Impediments, and identified ATD Management strategies.

1.5.1.2 Results

The main objective of this study was to elucidate and contribute to an extended knowledge base in the research area of ATD and to build a collective platform for future research. The contributions of this study are both in the academic and practical aspects. First, the study shows that there is no one unified and overarching description or interpretation for ATD, and, therefore, we provide a 'state-of-the-art' review of significant issues which identifies aspects of previous studies, and examines how these studies have been conducted. This study also provides a novel descriptive model which 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. This will allow practitioners and researchers to use this model to assess and recognize what problems might occur while dealing with ATD and the consequences of these challenges being left unattended. This study also shows that there is a compelling need for supporting tools and methods for system monitoring and evaluation of ATD, but also shows that no software tools covering the full spectrum of ATD are yet available. Furthermore, the results demonstrate that maintenance and evolvability are the main challenges within ATD, due to the fact that all of the ATD challenges are related to compromises in these quality attributes. This paper highlights 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. Consequently, this paper provides several key dimensions that need to be taken into consideration when defining a refactoring strategy for ATD issues. In addition, this study demonstrates, to both practitioners and academics, the relevance of paying more attention and effort to remediate ATD during the software lifecycle, in order to decrease the level of negative impact due to ATD on daily software development work.

Figure 6 answers RQ1 by illustrating the most significant characteristics of ATD, addressing the relevance of ATD, different categories of ATD, impediments of ATD in terms of both challenges and negative effects, and the management characteristics of ATD with regards to different management activities, available methods/tools, and, finally, a focus on the different refactory aspects.

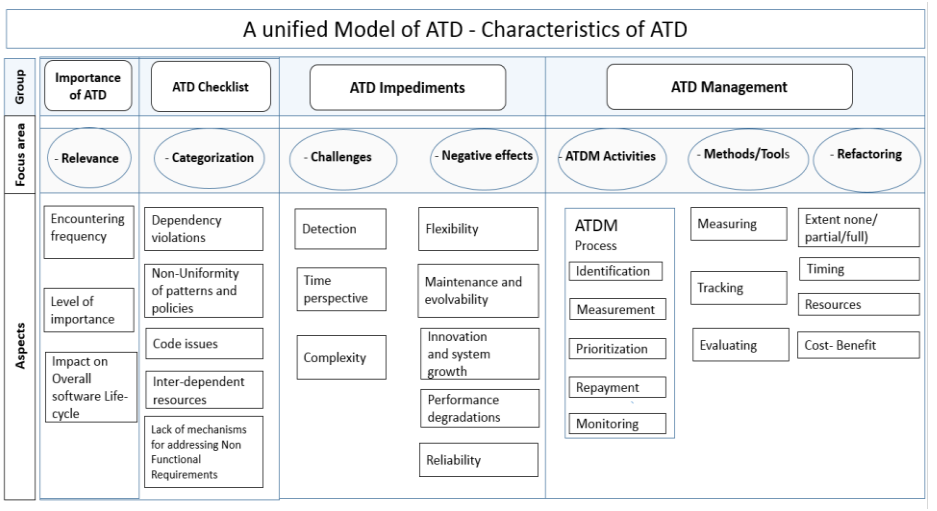

Figure 6: Characteristics of ATD, based on the results from Paper A.

1.5.2 Paper B: Time to Pay Up - Technical Debt from a Software Quality Perspective

The following sections will briefly describe Paper B. More details of this study are reported in Chapter 3.

1.5.2.1 Study Summary

The results of the SLR showed that, within the reviewed publications, the most frequently identified negative effects caused by ATD were compromises of maintainability and evolvability, which led to this second study, where we empirically investigated in what way TD affects different software quality attributes.

The second research question (RQ2) in this thesis addresses how TD affects different software quality attributes. In order to be able to answer RQ2, we conducted a study with the aim of understanding which quality issues have the most negative impact on the software development lifecycle process, and to determine the association of these quality issues in relation to the age of the software and to relate each of these quality issues to the impact of different TD types. This study was conducted through a combination of qualitative and quantitative research approaches and was conducted in three different stages.

First, we group-interviewed 43 software practitioners, with the goal of understanding which of the quality attributes were the most negatively affected by having TD. This stage also included an assessment of compromised quality attributes by an in-depth analysis, examining nine different TD issues and evaluating the impact each of these had on different quality attributes listed by ISO/IEC. In the second stage, an online web-survey was used, providing quantitative data from 258 software participants. In the third stage,

we conducted seven semi-structured follow-up group interviews with in total 32 industrial software practitioners.

1.5.2.2 Results

The second research question (RQ2) in this thesis addresses how TD affects different software quality attributes, which is addressed in Paper B. This paper provides an empirically based study on how practitioners experience and perceive TD, in terms of compromised quality attributes and their relation to wasted working time, based on both quantitative and qualitative data.

First, the results of this study show that practitioners identified *maintenance difficulties*, a *limited ability to add new features*, *restricted reusability*, *poor reliability*, and *performance degradation* issues as the quality attributes having the most negative effect on the software development lifecycle. When analysing these five quality attributes, the summary statistics from the survey showed that 60% of all respondents frequently or very frequently encounter maintenance difficulties during the software lifecycle, 45% of the respondents frequently or very frequently encounter restricted reusability, and 39% of the respondents frequently or very frequently encounter a limited ability to add new features.

Secondly, we found no evidence for the generally held opinion that maintenance complications increase with the age of the software. When studying how the different quality attributes examined were compromised with respect to the age of the software, our results showed that there were only significant differences in how the respondents encountered maintenance difficulties regarding its age. Furthermore, our results showed that, for software that is more than 20 years old and for systems within the age interval of 2-10 years, the respondents encounter maintenance difficulties most frequently, and, for systems within the interval of 10-20 years, a limited ability to add new features is the most frequently encountered problem. Respondents who reported restricted reusability as the most frequently encountered quality issue had software with an average age of less than 2 years. This study thus could not confirm the generally held view that the amount of compromised quality attributes increases with system age, but our results imply that it is important to remediate TD very early in the lifecycle in order to the keep the frequency of compromised quality attributes down.

Thirdly, we show that TD affects not only software productivity (in terms of wasted time) but also that several quality attributes of the system were negatively affected by TD. The results showed that there is a significant positive linear correlation between the frequency of encountering all of the investigated quality issues and the estimated amount of wasted time. The strongest relationship was found between the frequency of encountering poor reliability and wasted time, followed by a limited ability to add new features and maintenance difficulties.

These findings highlight the importance of understanding how TD negatively affects the overall system quality, in order to proactively manage it in terms of allocating time, resources and additional effort. These findings provide strong empirical confirmation that both practitioners and academics need to focus more attention and effort on deliberately remediating TD, in order to reduce future costly interest payments.

1.5.3 Paper C: The Pricey Bill of Technical Debt - When and by whom will it be paid?

The following sections will briefly describe Paper C. More details of this study are reported in Chapter 4.

1.5.3.1 Study Summary

From Paper A, it was evident that limited knowledge and few supporting tools are available to measure the extent of TD within a software application, and, in addition, the time spent on TD related issues is not made explicitly visible and measurable. The lack of this knowledge can result in the fact that software development organizations are not aware of the interest that they are paying on TD, and therefore they might not give TD management necessary attention.

The third research question (RQ3.1) in this thesis addresses how much software practitioners *estimate* the negative impact on daily software development work due to TD. To answer RQ3.1 and with the result of the previous study (Paper B), addressing the negative impact TD has on different software quality attributes in mind, the goal of this study was to empirically investigate how software practitioners perceive and estimate the interest payment of TD. More specifically, the main goal of the study was to examine the amount of estimated wasted time caused by TD interest during the software lifecycle. The aim of this study was also to investigate what type of TD generates the most negative effect and the activities on which the extra time was spent as a result of experiencing TD. Furthermore, this study also examined the ways in which the age of the software system affected the wasted time, the frequency of encountering different TD types, and the frequency of having to perform the extra activities. Finally, this study also investigates the ways in which different professional software roles are affected by these artifacts.

To accomplish this goal, we conducted a study using a combination of qualitative and quantitative research approaches. First, a web survey to seven companies within our networks with an extensive range of software development was sent out, and invitations were also published in software engineering related networks on LinkedIn. In total, this survey returned 258 complete answers (completion rate of 83%). Secondly, we conducted follow-up interviews with 32 industrial software practitioners, who had all participated in the first survey. During the semi-structured interviews, the compiled results from the previous survey were presented to the interviewees, where some of the most interesting findings were highlighted together with questions related to the specific area of research. This presentation allowed the interviewees to more easily relate the interview questions to the results of the survey.

1.5.3.2 Results

The third research question (RQ3.1) in this thesis, attempts to explore how much software practitioners *estimate* the negative impact on daily software development work due to TD to be, which is addressed in Paper C. This study offers a contribution to TD research, with respect to the existing body of knowledge, in several respects. The single most striking

result emerging from this study is that, on average, software practitioners estimate that 36% of all software development time is wasted because of paying the interest due to TD. The result reveals that the majority of the wasted time is spent on understanding and/or measuring TD. When studying how different professional roles perceive TD, this result reveals that different roles are affected differently by TD. We found that different roles waste time on different activities, hence experiencing different negative impacts of TD. When examining if, and in what way, the amount of perceived wasted time varies with respect to the age of the software, this study shows that the degree of the wasted time does vary. Although the amount of wasted time result does not show a linear progression, the result shows that wasted time varies in relation to the system's age. The findings in this study contribute to new knowledge concerning TD, stressing the importance of the fact that organizations need to be aware of the amount of time and resources they are spending on their interest of TD and to deliberately focus on remediating their TD.

1.5.4 Paper D: Impact of Architectural Technical Debt on Daily Software Development Work - A Survey of Software Practitioners

The following sections will briefly describe Paper D. More details of this study are reported in Chapter 5.

1.5.4.1 Study Summary

An important and interesting result from the previous study (Paper C) showed that when software practitioners estimate the negative impact of several different types of TD, they perceive that ATD generates the most negative impact on their daily software development work (closely followed by Requirement TD).

The third research question (RQ3.2) in this thesis addresses if and how ATD specifically generates a negative impact on daily software development work. Based on the previous result and in order to answer RQ3.2, we conducted a more in-depth analysis of the architectural related aspects of TD, i.e., ATD.

The goal of this study was to examine how software practitioners perceive and estimate the impact of ATD during the software development process from several different aspects. The first goal set out to examine the level of negative effects ATD has on the daily software development work and compare this level with the negative effects of other types of TD. Secondly, we aimed to understand if the level of the negative effects due to ATD correlates with the estimated wasted development time during the software lifecycle. Thirdly, the negative effects due to ATD is commonly believed to have an increasingly negative impact with respect to the age of the software. Finally, this study aimed to assess the extent to which the level of negative effects due to ATD differs in relation to the age of the system and if different professional roles are affected differently by specific ATD.

The data in this study were collected via a survey. This paper is, to some extent, also related to our previous paper, Paper C, where we study and compare several different TD types. However, even if the data are collected using the same survey, this study focuses on the architectural aspects of TD and does not focus on other types of TD besides ATD.

By focusing specifically on ATD, this means that we can provide a more in-depth analysis of the architecturally related issues of TD and provide more detailed statistical analysis of the data.

1.5.4.2 Results

The fourth research question (RQ3.2) in this thesis concentrates on ATD specifically and addresses the ways in which ATD produces a negative impact on daily software development work. This question is answered in Paper D, in which we study the negative impact ATD has on the overall daily software development work, with respect to wasted working time and additional activities performed.

First, the results of this study show that practitioners experience that ATD has the highest negative impact on daily software development work. The results of the study also show that the level of negative impact due to ATD is introduced early, and thereafter remains during the entire software lifecycle. Based on evidence from our survey, this study does not support the currently held belief that the negative effects due to ATD increase with respect to the age of the system. This study also provides new insights into ATD research by showing that, despite the different responsibilities and working tasks of software professionals, ATD negatively affects all roles without any significant difference between these roles. This study contributes to an empirical confirmation that software companies need to invest in continuous refactoring from the conception of the system in order to maintain the negative effect generated by ATD at a future low level.

1.5.5 Paper E: Technical Debt Cripples Software Developer Productivity - A longitudinal study on developers' daily software development work

The following sections will briefly describe Paper E. More details of this study are reported in Chapter 6.

1.5.5.1 Study Summary

In the previously presented paper, Paper C, the results show that software developers *estimate* that, on average, they waste 36% of their software development time due to experiencing TD. Even if the respondents in that study were experienced in software development, and their estimates were likely to be formed by what they have heard, observed, and experienced at their workplaces, we were intrigued by the idea of conducting an additional study where the wasted time could be studied by using *reported* data instead of single occurrence based on *perception* and *estimates*. In order to answer RQ3.3, we extended the previous research exploration by incorporating a longitudinal study where 43 software developers, twice a week for seven weeks, *reported* their experience and interest due to TD. The goal of this study was to explore the negative consequences of TD in terms of wasted software development time. This study also investigates on which additional activities this wasted time was spent and whether different types of TD impacted the wasted time differently. This study also set out to

examine the benefits of tracking and communicating the amount of wasted time, from both a developer's and manager's perspective.

This study was initially presented and discussed during a workshop with several industrial companies within our network. Thereafter, the study was conducted using three steps, where the first step was a start-up survey gathering descriptive statistics about the respondents, the second step collected reported data from the respondents over 14 survey occasions, and the final step was a follow-up survey in order to collect retrospective specific data from each respondent. The result was also verified using supplementary qualitative semi-structured interviews with 16 participating respondents.

1.5.5.2 Results

The third research question (RQ3.3) in this thesis concentrates on the reported (not the estimated) amount of wasted software development time, in terms of software development productivity. This research question is examined in Paper E, where the result is based on a longitudinal study of reported data and interviews with software practitioners. This study makes a novel contribution to the TD research, where the analysis of the reported wasted time in this study revealed that developers report that they waste, on average, 23% of their software development time due to TD and that the wasted time is most commonly spent on performing additional testing, followed by conducting additional source code analysis and performing additional refactoring. The results also reveal that, on a quarter of the occasions where developers encounter TD, they are forced to introduce additional TD due to the already existing TD.

By studying the tracking process of the wasted time, it was apparent that none of the examined companies in the study tracked or measured the amount of wasted time due to TD, and none of the companies had an aligned strategy for addressing the interest of TD. In addition, this study shows that both developers and managers clearly see the benefits of tracking the amount of wasted time, but both professions are somewhat reluctant to implement such measures in practice. This "unwillingness" is recognized as a challenge for the companies.

1.5.6 Paper F: Looking for Peace of Mind? Manage your (Technical) Debt - An Exploratory Field Study

The following sections will briefly describe Paper F. More details of this study are reported in Chapter 7.

1.5.6.1 Study Summary

The previously described studies in Papers A, B, C, D, and E demonstrate the negative consequences of TD and/or ATD, from both a technical and/or an economic perspective. However, TD can also affect developers' psychological states and morale, which is the focus in RQ3.4. Drawing on the previous literature on morale, this study explores the influence of TD and its management on three dimensions of morale: affective, future/goal,

and interpersonal antecedents. In this study, we followed a mixed-methods approach to both quantitative and qualitative research. The quantitative approach was performed through a survey with 33 software developers, and the qualitative part of the study was conducted through eight semi-structured interviews.

1.5.6.2 Results

The third research question (RQ3.4) in this thesis addresses how TD influences developers' morale, and is addressed in Paper F. This study has several contributions to both software engineering research and practice. The study specifically concentrates on investigating the influence of TD on developers' morale, while the study is based on previous literature on morale, we introduce a novel approach for studying morale within the software engineering discipline.

The results from this study show that that the occurrence of TD can reduce developers' morale, where this can be described in terms of the fact that the presence of TD hinders developers from performing their tasks and achieving their goals. The results further show that the proper management of TD increases developers' morale, where the TD management can have a positive influence on all the different dimensions of morale since it is associated with positive feelings and interpersonal feedback as well as a sense of progress.

1.5.7 Paper G: Technical Debt Management: Current State of Practice

The following sections will briefly describe Paper G. More details of this study are reported in Chapter 8.

1.5.7.1 Study Summary

As previously shown in all the publications mentioned above, TD has a negative impact on software development from various different perspectives, and the results from these publications demonstrate the relevance of paying more attention and effort towards actively managing and remediating TD. Although a great deal of theoretical work on TD has recently been produced, its practical management lacks empirical studies. When implementing a TD management strategy, the *tracking* of the TD is an important key activity. Therefore, this thesis' research question (RQ4), focuses on how companies start tracking TD and the initial benefits and challenges of the tracking process. This study was conducted using both qualitative and quantitative methods. First, we conducted a survey of 226 respondents from 15 organizations and followed up with multiple case-studies at three companies which have started tracking TD. The case study included 13 semi-structured interviews and collection of 79 TD-related documents.

1.5.7.2 Results

The fourth research question (RQ4) in this thesis examines how software developing companies track TD and what the initial benefits and challenges are when introducing and

starting a tracking process. This research question is answered in Paper G, which investigates the state of practice in several companies in order to understand how these companies start tracking TD. The results from this study show that software practitioners estimate that, on average, they spend a substantial amount of their working time trying to manage TD (25%) and only a few of them have started tracking TD, where 7.2% of them apply a systematic tracking process in this regard. The results further show that the major reasons for this noticeably low proportion of companies having an implemented TD tracking process are due to lack of knowledge of what is necessary to implement it in terms of tools and processes, as well as a lack of awareness of what the negative effects of TD are before they occur. In order to help the initialization process for TD tracking, we propose a Strategic Adoption Model (SAMTTD). This model can be used by practitioners to assess their TD tracking process and to plan their next steps.

1.6 Future Research

The work presented in this thesis has its main focus on studying and understanding *in what way* and *to what extent*, TD in general and ATD specifically, influence today's software development work. Based on the synthesis of the results from this thesis' conducted studies, several different opportunities for future research are provided.

In these studies, we have explored the negative effects of experiencing TD and ATD, and, in future research, we plan to investigate a range of solutions based on the insights arising from this study in order to mitigate the negative impact of both TD and ATD.

As illustrated in Figure 7, we have outlined four main tracks of future research, covering both TD in general and ATD more specifically.

The first track's main focus is to provide support for the awareness and understanding of the causes of TD and ATD and thereby contribute to an improved *prevention and limitation mechanism* of introducing new TD in the first place. This could be performed by studying how different software development guidelines and practices contributes to a reduction of TD and ATD.

Software development companies depend on their software practitioners, such as developers, testers, and architects, to implement the requested software. Our research results show that it is important that practitioners are aware of TD and its negative consequences and thereby also actively work with refactorings and avoidance of introducing new TD in the software. The second track is related to the first track but has a more solution-oriented focus, where different approaches to practically *encourage and motivate* software practitioners to keep the TD at bay are offered. In this track, we will investigate if, and how, companies can implement a rewarding system to encourage practitioners on an individual level or a team or project level to avoid introducing new TD and rewarding them when reducing the already existing TD by conducting refactoring activities.

The third track addresses *remediation and refactoring strategies* in terms of investigating to what extent different types of remediation and refactoring initiatives can potentially have a positive effect on the negative effects of TD and ATD.

The fourth track aims to understand how TD and ATD remediation and refactoring activities are *prioritized* in relation to features and bugs in the agile backlog during software development sprints and to understand which different artifacts have an impact on this prioritization process.

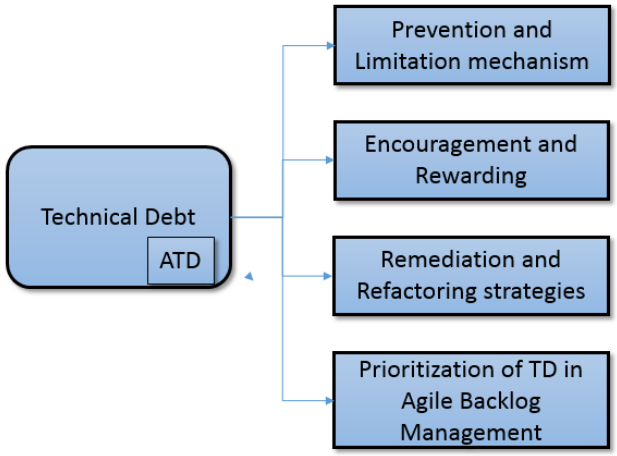

Figure 7: Possible directions for future research.

1.7 Conclusion

Returning to the goal of this thesis posed at the beginning of this chapter, it is now possible, based on the empirical data, to state the negative effects TD and ATD have on software development from several different aspects, encompassing technical, financial and social perspectives.

This thesis' study has a focus on investigating both TD in general and ATD more specifically, and when studying and comparing different types of TD, this thesis shows that ATD is of very high importance to software companies and that, among the different types of TD, ATD is the most commonly encountered type of TD. Furthermore, this study shows that, compared to all types of TD, ATD has the greatest negative impact on the daily software development work, estimated by all the different software professional roles surveyed.

Most commonly in the available academic literature, the negative effects due to TD are described in terms of maintenance complications and evolvability (limited ability to add new features) issues of the software. However, this study has been able to show that TD also has negative effects on software development from many other perspectives, which are also important and need to be addressed. The results in this thesis show that, in addition to causing maintainability complications and a limited ability to add new features, TD also has a negative effect on several other software quality attributes, such as a restricted reusability, poor reliability, and performance degradations, by providing a quantified estimation of the negative impact of each of the compromised quality attributes. Furthermore, this thesis broadly supports the common view presented in academic literature in this area, where our study empirically shows that almost all the investigated

types of TD cause maintenance difficulties as the most frequently encountered quality issue.When studying the frequency of encountering maintenance complications with respect to the age of the software, this study did find not any evidence for the generally held belief that the maintenance complications increase with the age of the software.

This Licentiate thesis also shows that software encountering TD caused software practitioners to perform additional time-consuming work activities. The time spent on these activities is referred to as *wasted time* in this thesis, and, by empirically assessing the amount of this wasted time, using both estimated and reported numbers by software practitioners, this study shows that they expend an extensive part of their software development working time on these activities. One striking result emerging from this study is that, on average, software practitioners (from several different roles) *estimated* that 36% of all software development time is wasted because of paying the interest due to TD. Furthermore, the results show that developers specifically *report* that they waste, on average, 23% of their software development time due to TD. The study also indicates that this wasted time negatively affects the development productivity and viability of the software. When studying on which different additional working task this wasted time is spent, the following activities were identified: performing additional testing, conducting additional source code analysis and performing additional refactoring.

This study also found that all different software professional roles are affected by TD. The results also reveal that, on a quarter of the occasions where developers encounter TD, they are forced to introduce additional TD due to the already existing TD. This burden of being forced to introduce additional TD demonstrates the contagiousness of TD, inferring that the interest cost of the TD might potentially grow exponentially.

When studying how TD affects the developers' morale, it is evident that working with software experiencing TD can reduce their morale, which can be described in terms of the issue that the TD hinders them from performing their tasks and achieving their goals. On the other hand, the results clearly suggest that proper management of TD increases developers' morale.

One key necessity to reducing the potentially negative effects of TD, is to track it. However, this study revealed that only 7% of the investigated companies had applied a systematic tracking process for TD. One reason for this significantly low number of companies who have implemented such a tracking strategy for TD can be described in terms of a lack of awareness of what the negative effects of TD are before they occur. Another reason is that companies have difficulties in finding supporting tools and methods to follow and also difficulties in understanding how different types of TD affect the software, thereby creating problems when prioritizing among them when planning for refactoring.

The overall results of this thesis empirically demonstrate that software encountering TD in general, and ATD specifically, causes several different negative effects, from both the technical, financial and social perspectives. The findings show that software development organizations need to understand and deliberately refactor TD and ATD in both newer projects and in more mature software. The findings in this thesis further demonstrate that the consequences of TD can, over time, result in issues such as project delays, software

quality complications, high defect rates, reduced developer morale and very low developer productivity. In the long run and left unchecked, these issues can seriously impede organizations' ability to innovate and grow by impeding innovation and expansion of their software. The findings indicate that software companies need to be armed with strategies and proactive management to enable them to track and manage the interest of TD. Such strategies could result in better, more informed decisions to balance the accumulation and the repayment of TD.