The impact of technical debt on stress

A case study of a large Swedish company

Simon Andersson

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

This thesis is submitted to the Faculty of Computing at Blekinge Institute of Technology in partial fulfillment of the requirements for the bachelor's degree in software engineering. The thesis is equivalent to 10 weeks of full-time studies.

Contact Information:
Author:
Simon Andersson
E-mail: siaq19@student.bth.se

University advisor:
Dr. Emil Alégroth
Dept. Computer Science & Engineering

Faculty of Computing
Blekinge Institute of Technology
SE–371 79 Karlskrona, Sweden

Internet : www.bth.se/didd
Phone : +46 455 38 50 00
Fax : +46 455 38 50 57

Abstract

Context: Technical debt, the process of introducing sub-optimal solutions for short-term profit at the expense of long-term effectiveness, is a new and upcoming research field and has been shown in multiple studies to affect different human aspects, such as moral and affective states.

Objectives: The aim of this study is to investigate how technical debt impacts stress in both developers and project managers.

Methods: The study consisted of two main parts, one literature review and one case study. The case study consisted of semi-structured interviews with 11 participants which were then thematically analyzed.

Results: The results show that technical debt which directly impacts the efficiency and performance of the developers has a negative impact on stress, which the study found to primarily consist of test debt and code debt. It also suggests that project managers are generally less impacted by technical debt and are more liberal towards introducing it to the system.

Conclusions: The study found that developers are more directly impacted by technical debt and are, hence, more critical towards its introduction. Project managers, on the other hand, are impacted by it to a lesser degree and therefore, in combination with the pressures and requirements by the customer, prefers a "good enough" solution over a perfect solution.

Keywords: Technical debt, stress, Human factors, Developer attitude

Acknowledgments

I would like to offer my special thanks to Dr. Emil Alégroth for providing the study with valuable insights, suggestions, and feedback. I would also like to thank the interviewees in the study for participating and providing the interview with honest and detailed answers.

Contents

Abstract i

Acknowledgments ii

1 Introduction 1

2 Related Work 2
2.1 Technical Debt . . . 2
2.1.1 Technical Debt Items . . . 2
2.1.2 Technical Debt Cause . . . 4
2.2 Technical Debt and Soft Factors . . . 4
2.3 Software Engineering and Stress . . . 5
2.4 Summary . . . 6

3 Methodology 7
3.1 Research Objective . . . 8
3.1.1 Motive and Research Questions . . . 8
3.2 Research Design . . . 9
3.2.1 Brief Search . . . 9
3.2.2 Summarizing . . . 11
3.2.3 Conducting the Review . . . 11
3.3 Interview Design . . . 12
3.3.1 Participants . . . 12
3.3.2 Delimitation of the Study . . . 13
3.3.3 Interview Structure . . . 14
3.3.4 The Interviews . . . 15
3.3.5 Analysis . . . 16
3.3.6 Create Initial Codebook . . . 16
3.3.7 Encode the Interviews . . . 17
3.3.8 Organizing Codes into Themes . . . 18
3.3.9 Analyzing the Themes . . . 20

4 Results and Analysis **21**
4.1 Technical Debt and Stress 21
4.2 Differences between Developers and Managers 25
4.3 Attitude towards Technical Debt 26
4.3.1 Developers on Technical Debt 26
4.3.2 Managers on Technical Debt 26
4.3.3 Attitude towards Eachother 27

5 Discussion **29**
5.1 Threats to Validity 30

6 Conclusions and Future Work **32**

References **34**

7 Appendix **39**

Chapter 1

Introduction

Software development is a complex process that often involves many components and people [38]. The constant balancing act between the interests of business managers, customers, and developers may introduce a variety of problems, one of which is technical debt (TD) [33]. TD is a phenomenon in software engineering inspired by economic concepts such as debt and interest. In practice, the concept manifests when sub-optimal solutions are introduced into a project [22]. These sub-optimal solutions may require developers to invest additional resources to achieve the same result—TD is associated with ongoing unnecessary expenditures [40, 45]. The cost of refactoring a sub-optimal system to make it optimal, e.g. reduce or remove the delta between the two, is generally referred to as interest. If this interest is not paid off in time, it may cause negative consequences for the project.

TD is not restricted to source code development, it can manifest in many parts of the system or its environment. Examples include overly complex architectural design, missing or badly designed tests, or inadequate build solutions [2].

Software engineering is a multi-faceted research discipline, drawing inspiration and knowledge from multiple disciplines, such as economics and psychology. A growing area of interest, due to the human component of software development, is psychology-based software engineering, also referred to as Behavior Software Engineering (BSE) [23]. In BSE, human factors such as motivation, attitude and stress are studied. Among other results, stress has been shown to have negative effects on effective software development.

This study takes a BSE-focused view on TD and aims to investigate if TD is a cause of stress and how that effect differs between developers and project managers. The results of this work are of general interest to both industry and academia, indicating that TD is a stressor that causes both immediate and chronic stress.

Chapter 2

Related Work

This section will present background information and relevant research on technical debt (TD), human factors in software engineering related to TD, and stress in software engineering. It also presents information and concepts important for the reader to understand in future sections of the report.

2.1 Technical Debt

TD is, according to its original definition by W. Cunningham, a process where developers deliver immature and sub-optimal code to speed up development at the expense of long-term productivity [10]. Since then, the term has evolved to include many different flaws and non-optimal solutions in projects [22]. A modern definition of software debt by Kruchten et al [22] describes it as "the invisible result of past decisions about software that negatively affect its future".

2.1.1 Technical Debt Items

There are, according to Alvares et al [2], 13 main types of TD, which are summarized in Table 2.1. Code smells are another related phenomenon in research papers, which the original author Fowoler [12], described as “we have learned to look for certain structures in the code that suggest (sometimes they scream for) the possibility of refactoring”. In essence, code smells are a parallel system to describe debt in the code and can be categorized as either code debt or design debt, depending on the smell. Examples of code smells are duplicated code, long methods, and large classes [12].

Debt item	**Definition**
Architecture Debt	This refers to sub-optimal implementations of the architecture of software, for example, lack of modularisation.
Build Debt	This refers to sub-optimal or unstable build-processes of software, for example, unnecessary code, build errors, or dependency warnings.
Code Debt	This Refers to sub-optimal code, for example, implementations or naming, that negatively impacts the software or its maintainability
Defect Debt	This refers to any known issues with the software, such as bugs, that should be fixed, but have not been yet.
Design Debt	This refers to violations of good code design, for example, low-coupling in object-oriented programming.
Documentation Debt	This refers to deficiencies in the documentation of a project, internal or external.
Infrastructure Debt	This refers to problems with the infrastructure in the organization that can impact development,
People Debt	This refers to issues regarding personnel, for example too few contracted employees or an unfavorable distribution of knowledge.
Process Debt	This refers to sub-optimal processes when developing software.
Requirement Debt	This refers to implementations of software that does not satisfy the given requirements.
Service Debt	This refers to deficiencies in the running and operation of a service.
Test Automation Debt	This refers to a lack of automation of tests in a project..
Test Debt	This refers to issues regarding tests in a project, for example, test coverage or low-quality tests.

Table 2.1: Technical debt items according to Alves et al. [2]

2.1.2 Technical Debt Cause

TD is a natural part of software development, especially in agile development where features and objectives might change substantially and where TD controls are often lacking [39]. The long term effects of unmanaged TD might vary based on the type of project but often lead to longer development cycles, higher costs, and customer dissatisfaction [40, 45]. Many developers cite low organizational interests and the prioritization of short-term goals as the main reasons for the debt in their projects [13]. A case study by Yli-Huumo et al. [45] found that the main causes of introducing TD were time pressure and code complexity, where the complexity and rigidity in the code made it too expensive to implement the optimal solution. Similarly, a study by Rios et al. [32] cited deadlines, lack of a well-defined process, and inappropriate planning as the most common causes of TD. A replication study of that study was conducted in Serbia which found the most commonly cited causes for technical deb to be deadlines, lack of experience, and ineffective project management [30]. Ernst et al, [11] reported that only 10% of the developers in that study had a manager that actively managed debt.

2.2 Technical Debt and Soft Factors

Software development is closely tied to the human aspects of the developers practicing it [23]. This field of research in software engineering is known as behavioral software engineering (BSE) and aims to improve the software development process by analyzing and reflecting upon human aspects. Many areas have been studied within BSE, for example personality [9], morale [5] and stress [15]. Research on emotional states suggests a correlation between positive emotions and increased productivity, creativity, and cooperation, traits which are presumably highly regarded in software development [24, 4].

TD affects both the developer's self-worth and their future goals and the magnitude of the effect was largely tied to how much it interfered with their effectiveness [41]. A study by Besker et al. [6] found that managing TD has a positive effect on the developer's morale, though the type of debt paid back impacts the effects. The payback of architectural and test debt was seen as positive by all respondents while other debt, such as management of documentation debt, was seen by some developers as a waste of time. Another study by Tom et al. [41] found that avoiding to repay TD, i.e. the interest, in the short term increased morale in some developers as it allowed them to get work done while it worsened the morale of others, especially people who are serious about code quality. Unmanaged TD had, in the long term, a clear negative impact on the morale of the developers.

There have also been some studies conducted on the relationship between other

psychological factors than morale, such as developer's emotional states, for example the one conducted by Graziotin et al. [18] who found that a lack of happiness in the workplace by developers has been linked to increased stress, depression, withdrawal, and lower productivity. A study by Olsson et al. [27] concluded not only that different smells impact developers differently but that some smells have a positive impact. It showed that cyclic dependencies had the clearest negative impact on the developers both impacting them negatively in terms of displeasure and having control of the situation but also their ability to produce new code. Broken modularization had a positive effect on the developer's emotions and while it would be hard to argue that developers would prefer miss-designed code to correct code, the smell did not seem to interfere with the readability or understanding of the code as some of the other smells

The presence of persistent TD was found, by Borowa et al. [7], to be preceded by four main emotional states; fear, shame, lack of agency, and haste. The fear and shame aspects in particular create a vicious cycle where developers are ashamed and afraid to report the debt they have caused but at the same time fear the responsibility and consequences of starting over and fixing the debt. Lack of agency mostly applied to participants in large organizations where they felt that they did not have any influence over decisions made which led them to mindlessly follow orders The haste aspect especially caused the participants to underestimate the time required to complete different tasks. Another field study by Ghanbari [16] on the matter also connected fear and a sense of insecurity when introducing TD into a system. Participants working with a system containing a large amount of debt reported similar effects on emotions, such as lowered confidence and frustration. Frustration was primarily seen when the debt was caused by external teams or developers as it was seen as time-consuming and a hindrance to their progress. The report also affirmed the belief that repaying TD was connected with positive emotions such as self-worth and a sense of achievement.

2.3 Software Engineering and Stress

Stress is a reaction to a demand, a stressor, with an influential outcome or consequence [28]. Distress, a negative reaction to stressfull situations might induce several symptoms such as anxiety, sadness, frustration and helplessness [21]. This, in software engineering terms, might translate to a developer becoming stressed by an upcoming deadline with the consequence of failing to deliver all demands on time [28]. Stress has been shown to impact the software development process negatively. It especially impacts the quality of the product as stressed developers introduce more errors to the code [1, 15] It has also been shown to negatively impact information sharing between developers [3]. According to a survey by Meier et al. [25] the agility of a project does not seem to have a strong correlation with the stress reported by developers and project managers. Developers had a

small preference, in regards to lowering stress, for more plan-based approaches while managers showed a preference for more agile approaches. Internal team communication, on the other hand, seemed to be the best indicator of stress. A study from India got a similar result where team communication and fear of obsolescence were the main causes of stress [29]. Another important factor for stress seems to be mood, where multiple studies find stress and depressive symptoms to be an effect of a bad mood and low motivation [18, 19, 26], side effects often linked to unmanaged TD [41, 27]. One study conducted on Japanese information system managers found that job overload and time-management had particularly large effects on the participant's stress levels [14].

Another similar study on information IS-managers from 1983 by Weiss [43] found that role ambiguity, lack of feedback, career development and organizational structure to be the most prevalent stressors. The stressors seemed to have the biggest impact on younger managers, presumably because older managers develop coping mechanisms to deal with the stressors and those who could not do that has changed their career-paths.

Software Project managers, though faced with many of the career and organisational issues of other managers find themselves, according to the study by Rivard et al. [34] coping with other types of stressors which is more prevalent. The study found that software project managers suffered from mainly two types of chronical stressors, task demand overload and goal conflict. Task demand overload meant that the managers were constantly overwhelmed by multiple and independent tasks that all required their immediate attention. Goal conflict in this context is described as the balancing act of discounted goals and requirements that all can not necessarily be fulfilled to a satisfying level, for example, requirements from different customers and internal processes. There were also mainly two acute stressors identified, loss of control and changes to the requirements.

2.4 Summary

Summing up all parts of the related work, section, it becomes obvious that psychological factors have an impact on software development. It is also obvious that TD impacts many psychological factors, some of which have been linked to stress. However, ther is a lack of empirical studies on how TD affects software developers, especially in regard to stress. Most studies about TD's impact on development have also been conducted from the perspective of developers which warrants research that includes a managerial perspective.

Chapter 3

Methodology

The study presented in this thesis includes two parts, one literature review, and one interview study. The study followed the structure outlined in Figure 3.1 where a research objective for the study was described. This objective was formulated before the literature study was conducted. The literature review of the study is presented in section 2 and the information gathered from the section is used to formulate research questions and validate the results. The interview study is presented in section 4. The following section will describe the methodology used for each part of the study.

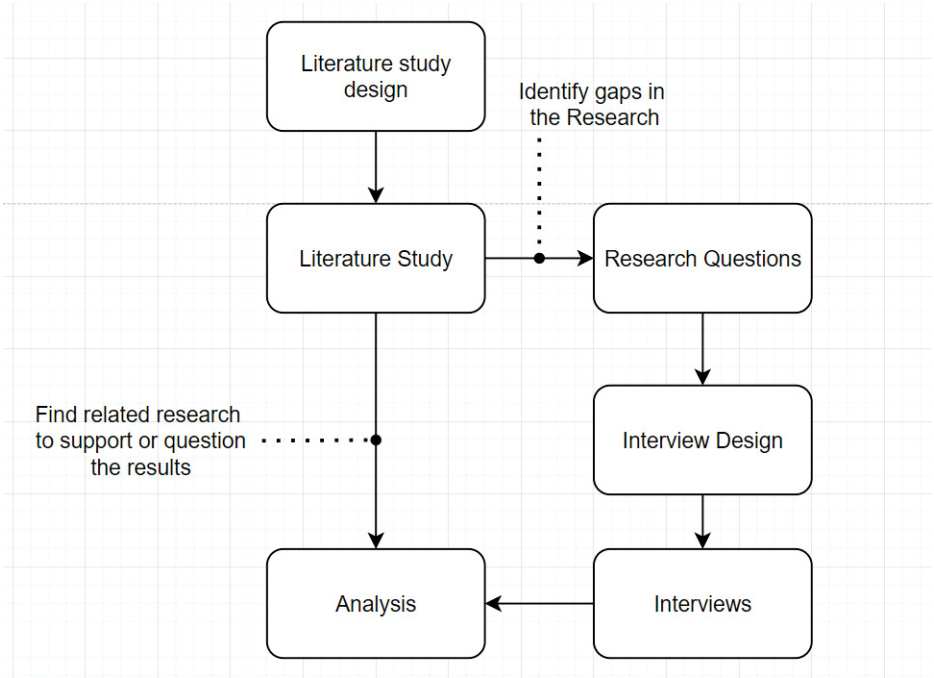

Figure 3.1: Methodology

3.1 Research Objective

The objective of the research was to study TD on human factors, however, the goal of the study was not clear in the start of the study, therefore a literature review was conducted to identify research gaps and formulate the specific research goal and research questions. A brief search on this area revealed opportunities for research pointed out by both Olsson et al. [27] "Much of the current literature on TD (TD) pays particular attention to technical or financial perspectives." and Besker et al [6] "The majority of previous studies focused on investigating the technical, financial, and organizational aspects of TD. However, until now, the literature has paid minimal attention to the human, and social aspects of TD, including the role of developers in the occurrence of TD as well as its consequences for them". The objectives of this research will be the following:

1. Identify gaps in the research in the field of technical debt and human factors
2. From these gaps of knowledge, construct research questions to further explore the subject
3. Construct an empirical study to investigate and answer the research questions.

3.1.1 Motive and Research Questions

The research on TD suggests that it leads to several negative long-term effects, including longer development cycles, higher costs, and customer dissatisfaction. [40, 45]. Human factors, commonly referred to as soft factors, have been studied in various forms in Software Engineering. TD has, in such studies, been shown to impact both morale and emotions [27, 41]. Considering the reported effects of TD, it would be reasonable to assume that it would have effects on stress. The research on stress and software development cite many causes, such as overtime and high pressure as the main contributors of stress [15, 31]. Currently, there does not exist any explicit research on the relationship between stress and TD, however.

Most research regarding TD is focused on the perspective of the developers. Some studies, notable one published by Rios et al [32] include project managers but the study did not focus on how those answers compared to answers from other roles. Including managers in the research could deepen the understanding of the relationship between both groups and identify opportunities for changes to increase efficiency and lower stress levels.

To minimize the amount of stress caused by TD, the connection between stress and TD must first be understood. Understanding of the concept is perceived to lead to better quality products and healthier and happier personnel as stress

has previously been shown to negatively impact both the quality of the software products [1] as well as correlate with symptoms of depression [19]. This will be explored by answering the following research-questions:

1. What is the effect of technical debt on stress?
This question will explore what effects TD has on stress levels. This can answer if TD has a direct impact, if any, on stress and if certain debt items have more impact than others as has been shown in other research exploring other soft factors related to TD.

2. What is the difference between the impact of technical debt on developers compared to project managers?
This question aims to explore if the stress-levels of project managers and developers are impacted differently by TD.

3. How does the attitude on technical debt differ between project managers and developers?
This question will explore if there are any differences in how managers and developers view TD. The answers might give insight into fundamental differences in how both roles view the problem and if their viewpoints clash.

3.2 Research Design

The literature review aims to identify gaps in academic research and find related research from which parallels and further conclusions can be drawn. Information from the Literature review will be used to formulate relevant research questions and construct relevant interviews as well as review the results from the interviews to support or question their validity.

The literature review will follow the basic principles outlined by Rowley et al. [36] which is similar to the to the "snowballing" guidelines outlined by Wohlin [44] with some extra steps, such as summarizing the literature. Wohlin's method also includes "forward Snowballing" where sources are found by investigating citations of the current studies, which this review did not utilize. It is, considering the number of sources the study found, unlikely that forward snowballing would have yielded any new relevant sources, and the methodology used in the study can be considered good enough for its purpose and should not impact its validity.

3.2.1 Brief Search

The first step of the literature review is to briefly search the available literature to familiarise with it and identify common terms and questions posed by it [36]. The review had an initial set of keywords provided by the University Advisor. A list of all these keywords can be found in Table 1. The brief search will mostly

consist of reading the abstract and summary of the studies to quickly get a grasp of its content. This will identify additional keywords that can be used for future research. Examples of effective keywords found is "stress", "emotion", "cause" and "presence". These keywords are used in Google Scholar to find articles satisfying the study's inclusion criteria, which can be found below. An effective search term might look like this:

"Technical debt" +morale

This instructs Google scholar to search for academic literature including the string "technical debt" and the word "morale". A search not including the citation marks would include research containing the words "technical", "debt" and "morale", including more irrelevant research in the process.

All searches were conducted using Google scholar which is a search engine focusing on academic articles, books, conference papers, and other academic literature from a wide array of databases and websites deemed academic [42]. This inevitably introduces a risk of missing out on articles not picked up by Scholar's search algorithms. The benefit is that it will not be limited to specific databases and therefore encompass a larger part of the available literature. Furthermore, the study utilizes citation pearl growing where it looks at references used in other studies which lowers the risk of missing important sources.

The review is limited to free, publicly available, research as well as research accessible through Blekinge Institute of Technology, BTH. The study used articles from the following databases and libraries:

- ACM Digital Library
- IEEE Xplore
- SpringerLink
- ScienceDirect
- MIT Libraries
- ArXiv
- Researchgate
- American Psychological Association
- J-Stage
- PeerJ
- JStor

3.2.2 Summarizing

This step intends on distilling each relevant source and writing a short summary about it, for example: "[11] found that only 10% of the developers had a manager that actively managed debt". Important sources relevant to the research will be marked with a special note to imply their importance for the next step.

The criteria for including a study in the review are the following:

1. **Length**
   The report must be published and available in full text
2. **Relevancy**
   The study or article must be relevant to the research topic of TD, soft factors and stress. A study identifying the effects of TD on morale, for example, is highly relevant while a study comparing machine-learning algorithms to identify TD is not as relevant to this study. As a general rule, a study or article was deemed relevant if it:
   - Studied the cause of TD
   - Studied the effects of TD, both in terms of business and soft factors.
   - Studied the effects of stress in software engineering. This criteria was added after the study had identified the gap in the literature.

   Some other literature was also included when needed, for example when explaining important expressions and terms.
3. **Validity**
   The source must use a suitable and reliable methodology and research methods, for example including trustworthy references to back up their claims and presenting a big enough sample size to support their conclusions.
4. **Language**
   The study or article must be written in English

3.2.3 Conducting the Review

The next step is to organize the sources into different categories based on their findings. From this, the basic structure of the review can be created. Each category can then be expanded by finding more sources. The new sources will primarily be found using citation pearl growing from the original sources found in the above steps. Citation pearl growing is a technique where new sources are found by exploring the references of existing sources [36]. Once a gap in the research had been identified, in this case stress, the scope of the review expanded to include studies relating to stress and software engineering. All new sources are

exposed to the same criteria as presented in Section 2. In total, the review found 34 sources that met the inclusion criteria.

3.3 Interview Design

The empirical part of the study is primarily exploratory as there exists only a small amount of research on TD and stress and no direct industrial study investigating their relationship. The study will be a case study following the structure outlined by [37] and consist of semi-structured interviews of both software developers and project managers. The interviews were classified as semi-structured since, although there was an interview guide, the order of questions was not strictly followed. In addition, due to the exploratory nature of the study, follow-up questions were expected to be required to go deeper into areas of interest. This style of the interview could be extra valuable in a case like this when certain terms, such as TD, can be interpreted differently by different people [11]. It allowed interviews to flow more naturally and uncover TDs that the participants originally did not recognize as TD, for example, documentation and people debt. It also allowed them to elaborate on problems in their project without explicitly calling it debt.

3.3.1 Participants

The participants of the study will be sampled from a large Swedish consulting company using convenience sampling [35]. Convenience sampling means that the study will include any eligible participant that meets the study's criteria. Robinson recommends convenience-based qualitative studies to limit their intended group or location to avoid unfounded generalizations towards the whole population, who in this case is all developers and managers. The criteria, which were applied before the interviews took place, were the following:

1. **The participants must have at least 2 years of experience with software projects.**
   This is to increase the chance that they have encountered and reflected on TD and have some industrial experience in software development.
2. **The participants must speak Swedish or English**
   The interviews will be conducted verbally and thus, the language barrier must be respected.
3. **No prior interviewee has worked on the same project**
   To avoid overrepresentation of certain projects in the final results, which might skew the results, all participants will be interviewed about different projects.

The study found 11 eligible participants for the study and is based upon the experience from many projects as many of the participants had worked with a plethora of projects in different roles. Table 3.1, presents a summary of all the interviewees' demographic data collected during the interviews. The small sample size might become a threat to the study's validity. Criteria 3 of the sampling criteria combined with qualitative interviews that investigates and explores each participant in-depth should hopefully minimize this risk.

This sampling inevitably limits the study's external validity and how much the findings of the study can be applied to the wider industry. The findings, however, can give hints and be the basis of further research on the subject in other contexts, which would increase its external validity. As described by Jimenez et al. [20], there is often a trade-off in studies in regard to internal and external validity where this study traded some external validity to increase the validity in the given context. As Jimnezes argues, internal validity is often a prerequisite to external validity and as this study is of an exploratory type, increased internal validity in the current context is deemed preferable.

ID	**Role(s)**	**Years of experience**	**Area**
1	Project Manager	28	Web development and system development
2	Developer	12	Web development and system development
3	Developer	6	DevOps and system development
4	Developer	4	Web development
5	Project Manager	20	Web Development
6	Developer	40	Web Development
7	Developer	2	Web development
8	Developer	3	Mobile app development
9	Developer	5	Security and system administration
10	Developer	3	Web development
11	Project Manager	28	Web development and system development

Table 3.1: List of interview participants

3.3.2 Delimitation of the Study

This study is limited to people working in a large Swedish consultant company. The participants primarily worked with developing web services in agile teams

in large companies or government agencies. Hence, companies that could all be considered mature organizations that should both follow some standards and have rigid development processes. The projects from these organizations were, as stated, sampled through convenience sampling since they were the main contractors of the consultant company from which interview subjects were drafted. There also existed a handful of internal projects within the consultant company itself.

3.3.3 Interview Structure

The interview will be structured as a pyramid as suggested by [37] where the first questions will be specific, asking the interviewees questions about their background, experience, types of projects they are working on, etc. The questions will then be loosened up and be more open-ended. This lets participants talk about TD from their perspective and how it influences them which is important as developers and software managers might experience debt differently.

All interviewees were informed before the interview that their answers would be anonymous. This is to protect the integrity of each participant and let them speak freely about their experiences without risking any repercussions from their employer.

Multiple interview questions were used to try to answer one research question as the different perspectives and angles might nuance the answers and overall increase the quality of the results. Each research question the interview-questions aims to answer is presented in the parenthesis after each question. The basic structure of the interview will consist of the following 11 questions and sub-questions—18 questions in total.

1. Describe your role in software development. What roles and types of projects have you been working on?
2. How long have you been working with software development?
3. On a scale of 1-10, where 1 is ignorant and 10 is very knowledgeable, how well do you know the concept of technical debt?
4. Have you or your team experienced technical debt as a problem?
5. How has technical debt been treated on the projects you have been working with?
    (a) Describe the technical debts that the project had (**RQ1, RQ2, RQ3**)
    (b) How prevalent were the technical debt and how did it impact you in your work? (**RQ1, RQ2**)

(c) How did the technical debt impact the negative stress in you and your team? (**RQ1, RQ2**)

(d) Would you primarily categorize the negative stress as an acute stressor or a chronic stressor? (**RQ3**)

(e) Did it exist any guidelines on how to deal with technical debt? (**RQ1, RQ2**)

(f) Was there an active work to reduce the technical debt? (**RQ1, RQ2**)

6. What are the most common causes of stress in your work situation? (**RQ3**)
7. Can you describe your approach to technical debt (e.g. zero tolerance, something that can not be avoided, etc)? (**RQ3**)
8. Do you feel that the developers / project manager have an understanding of your approach and work with technical debt? (**RQ3**)
9. If you have worked as both a developer and a project manager, how did you feel that your attitude towards technical debt changed? (**RQ3**)
10. Do you think that all stress created by technical debt is bad?(**RQ3**)
11. Do you have any other comments about technical debt and/or stress that you think we missed in the interview?
12. Do you know anyone else in another company who would be suitable to interview?

The interview questions were analyzed by the university advisor and tested in a pilot interview with an interviewee with some industrial experience in software development. The pilot interview did not result in any changes to the interview but ensured that the questions were logical and easy to understand for the interviewees.

3.3.4 The Interviews

The interviews were conducted and recorded both in-person and remotely, depending on the location of each participant. The interviews lasted between 15 and 40 minutes depending on the availability of the participant and the value and content of the conversation. The average length of the interviews was 22,5 minutes with a standard deviation of 8,5.

Each interview was then transcribed into a separate document, documents available in the replication package [1]. The interviews were transcribed word for word,

[1] `https://drive.google.com/drive/folders/1dhe3GI0DnKAsHlOdyDMZ_L9v-ZzZ6BcT?usp=sharing`

including pauses but without filler sound that does not bring any value to the actual answer. Each answer was also anonymized to protect the integrity of each participant. The following type of answers were anonymized:

- Naming a company or institution
- Naming a person
- Naming a project

In a few instances, parts of some interviews were cut upon the request of the interviewer. This was parts that revealed particularly sensitive information though it should not have an impact on the final result.

3.3.5 Analysis

The results gathered from the interviews will be analyzed using thematic analysis where themes will be identified and analyzed in the Qualitative data. A theme, though a loose concept, is a recurring and identifiable pattern in the dataset that can be used to draw conclusions [8]. A combination of how many times and how much a theme is brought up in different cases is an *indicator* of its importance though the context of the sampling frame is important. A specific sub-group of the sample might vastly over-represent certain viewpoints and thus inappropriately skewing the results for the whole sample group. According to Braun et al. [8] thematic coding is the preferred way to analyze data from case studies and interviews. The codebook has been validated by the university advisor and can be found, together with the interviews, in the replication-package [1]. This study used a Google sheet as the code book as it is free and easy to access for external sources to analyze. The study used the steps outlined in Figure 3.2.

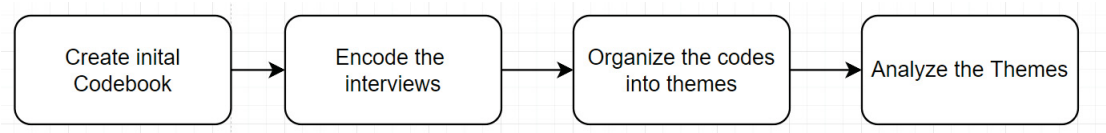

Figure 3.2: Interview encoding

3.3.6 Create Initial Codebook

The thematic code analysis will use data-driven coding, often called open coding, to code the interviews following the example established by Gibbs and Graham

[1] `https://drive.google.com/drive/folders/1dhe3GI0DnKAsHlOdyDMZ_L9v-ZzZ6BcT?usp=sharing`

[17]. This means that the coding starts with no, or very few pre-determined codes. Instead, codes are generated as new concepts of importance are identified throughout the interview transcripts. Each citation, or part of a citation, can be coded with multiple codes, i.e. a one-to-many relationship. The initial codes were written down in the sheet document. The initial codes for this study were the primary study areas and all the TD items, as shown in Figure 3.3. These are not necessarily meant to persist during the coding process but will make it easier to organize the first interviews and steer the coding into a logical pattern. Note that the first two codes, role and time, are only included for the demographic data and are not part of any theme.

A	B	C	D	E
Code	Answers			
Role				
Time:				
TD				
TD Items->Inflation				
TD Items-Architecture Debt				
TD Items->Documentation				
TD Items->Code Debt				
TD Items->Pople Debt				
TD Items->Infrastructure Debt				
TD Items->Defect Debt				
TD Items->Test Debt				
TD Items->Test Automation Test				
TD Items->Service Debt				
TD Items->Requirement Debt				
Stress				
Attitude				

Figure 3.3: Initial codebook

3.3.7 Encode the Interviews

With the initial codes in the codebook set up, the first four interviews were encoded. The answers will be put in the next empty column to the right of the corresponding code. An example of encoding is this answer from interviewee 1 on the question; "How much did the technical debt impact you in your work? You said earlier that it impacted you all the time but is that impact large compared to the rest of the work?"

"*No, that's not what drives our work..* **You have to look ahead too. It may**

be case that the deal is not located where the technical debt is. One can solve technical debt by making technology obsolete, right?"

The answer can be split into 3 parts which individually be put into different codes. The first part (italic) describes the severity of the TD, which is coded with the code "TD Presence". The second part (bold) reveals the interviewee's attitude toward TD and will therefore coded with the code "attitude towards TD". It could also be classified as a "Business" or "Economics" code. The last part (Underline) Could also be classified as an attitude, but is better described as the interviewee's view on how TD is removed, hence this is coded as "TD Removal".

Once the first interviews were coded, the existing codes were analyzed and reorganized. Examples of this would be duplicate codes and codes that contained the same or almost the same answers. Codes that have received a large amount of answers, for example "TD Effects" can be split up into smaller codes, for example "TD Effects->Development" when it impacted the participants' development and "TD Effects->Extra work" when it caused the participants extra work or forced them to work overtime. After this, all answers were removed from the codebook and all interviews were re-coded together with the rest of the interviews. In the end, the code book contained 53 codes.

3.3.8 Organizing Codes into Themes

The codes were arranged and organized into a hierarchical themes, where multiple sub-themes could be a part of one theme. A sub-theme is a more specified theme of another theme, for example, "cause of TD" is a sub-theme of "TD". Each sub-theme can, in this structure, contain further sub-themes which in the case of "cause of TD" includes "Time", "code-reviews", "structure" and "knowledge". The answers resulted in the following themes:

Technical debt

This is the largest theme containing 5 main-sub-themes. Effects of technical debt contain the reported effects of technical debt and how it impacted the interviewee and his or her team. "Presence of technical debt items" contains the reported types of debt in the projects. Cause of TD contains the reported causes for TD in the interviewee's project. "Amount of TD" represents how much TD a given project had and how it impacted development. "TD Removal" contains reported structures and workflows used by the participants to remove debt as well as their reported Guidelines to manage TD.

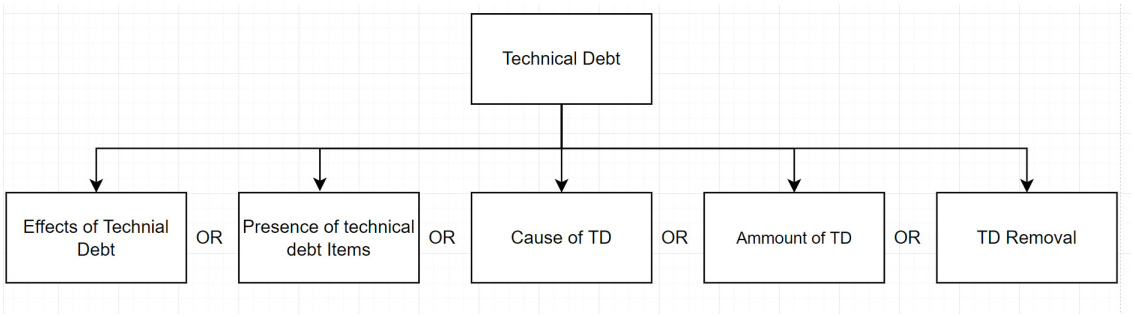

Figure 3.4: Technical debt theme

Attitude

This theme focuses on reporting the attitude of managers and developers towards the introduction of TD. "Understanding other roles" contain citations about how they view each other, for example, if the developers feel like the managers understand their situation and vice versa. "Towards TD", on the other hand, is the attitude against the concept of TD, in other words, how willing they are to introduce it and under which circumstances. "Towards stress" contains answers about both the developer's and manager's attitudes and relationships toward stress. The codes of each sub-theme, though not visualized in Figure 3.5, contain 2 separate codes, one for the attitude of managers and one for the attitude of developers. This was done to ease the analysis part of the themes.

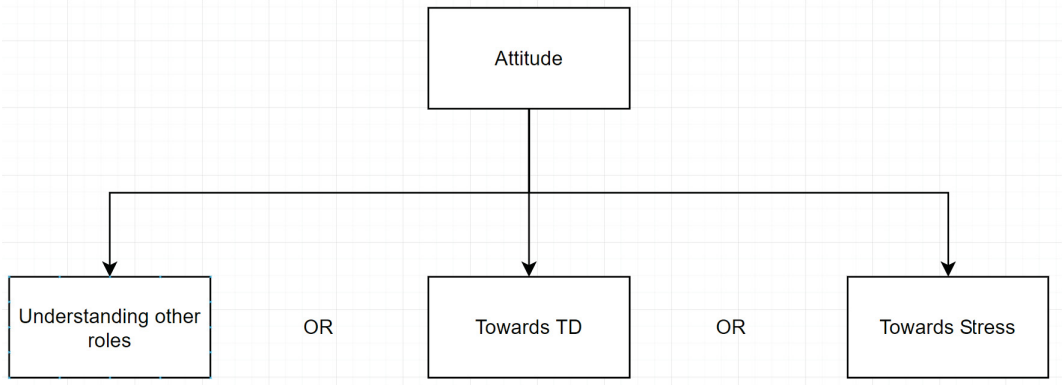

Figure 3.5: Attitude Theme

Stress

This theme contains all stress-related answers. The first sub-theme contains the stress caused by TD. The two sub-themes, chronic and acute, contain the answers which can be categorized as either a chronic or acute stressor. The "Factor of stress

caused by TD" contains the participant's estimate of how much of the stress, on average, is caused by TD. The stress Cause theme contains the main causes of stress and is not limited to stress caused by TD.

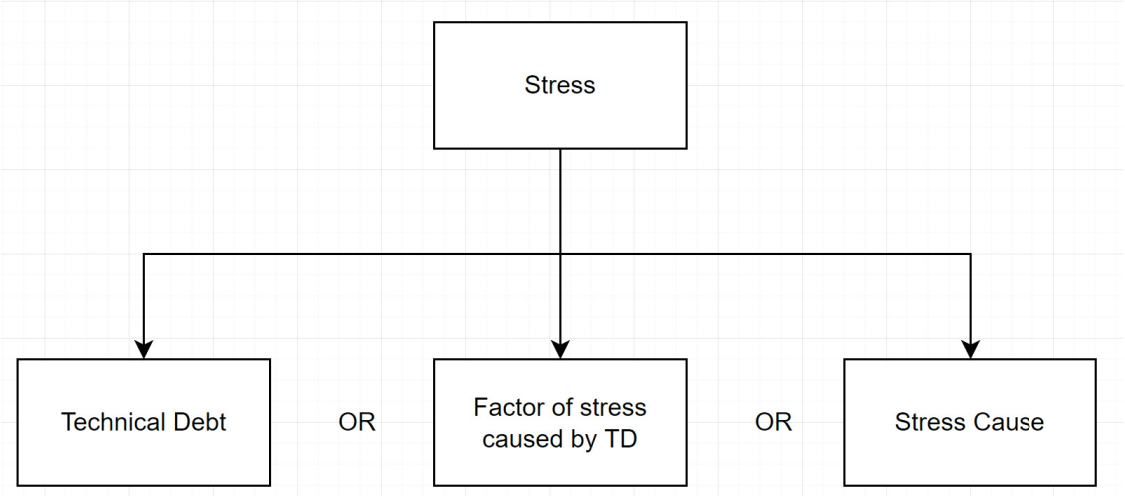

Figure 3.6: Stress theme

3.3.9 Analyzing the Themes

The key points of each theme will be presented and analyzed in the results and compared to similar sources to examine its validity. The analysis of the codes was solely conducted by the author and was not verified by an external source. The analysis will require that two or more themes are compared to each other to draw further conclusions, for example, if different debt types have different effects on the stress levels. To find these correlations, the following steps will be used:

1. The analysis will start by examining a given theme, for example how much stress TD causes. The results from that might suggest that some participants were particularly impacted while others were not impacted at all.
2. The codebook will then be used to search for commonalities of the impacted participants in other codes and themes. It would use the example above, to be reasonable to look at the TD theme and look if the participants are overrepresented in a certain sub-theme, for example, the effects of TD.
3. After the potential causes, or effects, are found, it is important to try and find contradicting evidence to minimize the risk of presenting false correlations. Using the previous example, one could examine why the others are not impacted. Do any of these share the same similarities? Is there enough quantitative data to support the statements?

Chapter 4

Results and Analysis

As presented by the method, the analysis ended up with three different major themes: Stress, technical debt, and attitude. The stress theme and technical debt themes will be used in conjunction with each other to answer the first two questions and the third theme will be used to answer the third research-question. Please note that the interviews were conducted in Swedish and the quotes will be, for the reader's convenience, translated to English. The original quotes can be found in the replication package[1].

4.1 Technical Debt and Stress

Investigating the stress caused by TD reveals that TD does impact, to varying degrees, the stress of both developers and project managers. Variance of perceived stress could be attributed to how often they got stressed, for example interviewee 11 who was highly impacted by stress: "the awareness that not all parts are redundant and can crash, it's grinding. It is there all the time, every day of the week, around the clock.". In contrast, interviewee 8 got stressed primarily in combination with deadlines: "I would probably say that it was more stress closer to deadlines. It's like a seasonal stress.".

The stress also varied in how much stress the participants suffered from, where interviewee 4 suffered from a lot of stress: "When you troubleshoot and troubleshoot and you fix one error and then you notice that there are just as many errors left, that is extremely stressful.". Contrarily, interviewee 2 did not get stressed at al: "No, I would not say it [TD] affects me.". A common answer from the interviewees was in the style of: "Actually, I think I should be [Stressed by TD], but stuff like that does not stress me anymore" (interviewee 1) or that "I am a person who does not get stressed" (interviewee 8).

[1] `https://drive.google.com/drive/folders/1dhe3GI0DnKAsHl0dyDMZ_L9v-ZzZ6BcT?usp=sharing`

There is a strong correlation between which type of stress, acute or chronic, TD causes to each individual based on how much stress they feel. Individuals who, as explained above, did not get that stressed by TD were, when stressed, primarily affected acutely by the side-effects of TD and not the presence of TD itself, for example: “No, it becomes an acute stress, for example when we are going to have a demo and my boss asks me “why does not this work”.Then you sit there and wonder if it’s you who have made a change that broke everything or if someone else in the team made a change that broke everything. Or if it’s a new bug.”. The stress that these individuals suffer does not seem to be the cause of the presence of TD itself, but its indirect consequences, such as unexpected problems interfering with deadlines perhaps caused by TD.

People who did state that they suffered stress from TD more often cited a chronic type of stress. It seems as though these people were more stressed by the debt itself and its direct consequences, for example, lack of tests or making it more difficult to implement new features. Interviewee 4 explains it as “one can often sit and ponder on these technical-debt things while receiving new requirements. It’s stuff like that that makes you work longer hours or think about it during the evenings and such when you are not working”.

Including the amount and type of TD in the analysis reveals that the amount of TD in a system somewhat correlates with the percentage of stress. Graph 4.1 is a rough visualization of the relationship between TD amount (estimated on a scale of 1-5) and the percentage of stress caused by TD, estimated when necessary. The stress axis represents how much TD contributed to the total stress amount and does not indicate the absolute stress levels of the participant. The TD amount scale was estimated on their answers in the interviews, where an answer 5 on the scale looks like: “Yes! I would say that. To the degree that we had to stop development to repay the technical debt” and “Yes, the whole system was pretty much technical debt”. A zero on the scale means the project had no TD. The graph, therefore, should not be seen as a generally accurate representation of reality but rather as an indication and visualization of the reported data.

The graph shows a trend where the amount of perceived TD seems to increase the amount of perceived stress. The spread of the results is large, however, which suggests that other factors are at play. Delving deeper into the results reveals that the type of debt appears to be a better determinator of stress compared to the sheer amount of debt, for example test-debt and code debt.

Lack of tests and test automation appears to be a direct cause of stress in software projects, especially when maintaining a project. It causes both acute stresses as described by interviewee 4: "When you troubleshoot and troubleshoot only to discover that there are just as many bugs left, that is extremely stressful." as well as interviewee 6: "It’s like, every little change you make, you don’t know if that’s going to make something else break." as well as chronic stress as pointed out by

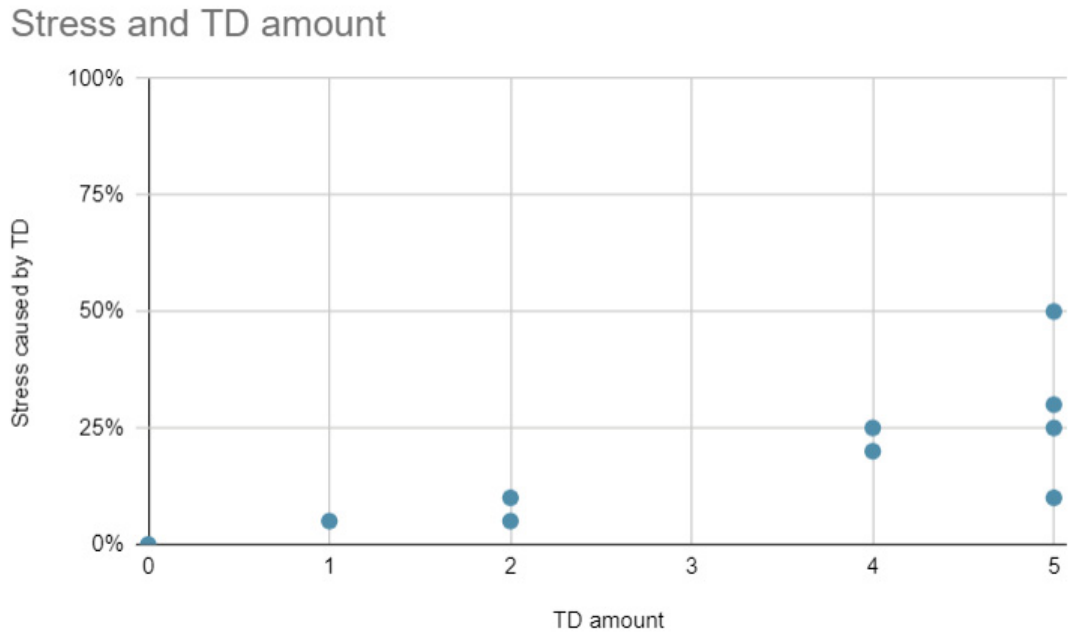

Figure 4.1: Graph representing stress and TD amount

interviewee 11: "When you know that certain parts of the system are less stable because we have not made them completely redundant". These statements are backed up by the fact that many developers attributed test-driven development as an effective way to reduce TD in the system, for example, interviewee 4 stated that "it [TDD] saved us". This is in line with other research on soft factors and TD, for example, Tom et al. [41] who found that it was mainly the TD that interfered with the participant's effectiveness and caused lowered morale and Besker et al. [6] who found that developers saw the payback of architectural and test-debt as the most crucial debt to pay back. There are tendencies to suggest that architectural debt is causing stress too. Interviewee 3, who was stressed by the TD, suffered from architectural debt: "And then it just happens to be the case that [company B (Company has been anonymized)] is very good at running microservices, but coupled, which means that you require all microservices, otherwise it does not work and everything just blows up.". It would not be unreasonable, considering the research by tom et al. [41] and Besker et al. [6], to assume architectural debt would have a similar impact on stress as test debt or code debt. The data in this study is limited in this regard however, as the architectural debt was not a commonly reported debt and the other participants who did suffer from it did not consider it particularly critical.

Code debt was the most common debt in the projects and was mostly related to outdated components and frameworks, for example, interviewee 4 stated: "and

secondly, we are working with a framework that had end-of-life in December". A supporting statement was given by interviewee 5: "There is not enough work done with maintenance. You know, update the frameworks we are working with to the newest versions. There is no continuous maintenance". This debt does not necessarily seem to cause any stress if it is located in external components: "There's quite a lot of debt located there [external components]. The problem is that, yes, the things that are there actually work. So who can afford to change it?". It seems to cause more stress when it directly interferes with the development, for example, interviewee 3: "I get assigned a new task or I think up a new idea to do but its not possible because the technical debt blocks me and that stresses me, I want to do it" or interviewee 4: "Every time you bumped into something that was a sketchy solution or that lacked test-coverage it became a stress-moment because you could not trust the code, or the architecture for that matter, to 100%".

One way in which TD had an impact on stress was combined with deadlines. Many interviewees stated deadlines as their primary cause for stress and the TD became stressful when it interfered with the delivery, for example interviewee 5 stated: "If you find things late in the project, then it's definitely going to impact stress levels: But if you already knew about it beforehand, then it's a different story". This is supported by interviewee 7 who said: "Especially if you have an upcoming deadline but do not understand how your going to implement different things and get the architecture to work because it is so complex". It would be reasonable, therefore, to assume that TD amplified the stress participants felt during releases. Especially as the interviewees without or with limited TD, (interviewees 2 and 8) did not mention time as a major cause of stress. This is backed up by prior research that found that TD caused longer and more unpredictable development cycles [45], which is presumably the main driver of stress in this situation. Some participants, like interviewee 10, actively assigned extra time when working with debt-filled parts of the system: "We know that it's going to take extra time to develop things in the older components...".

In summary and to answer the first research question, the results indicate that TD has an effect on stress. The impact of stress varies depending on the individual, the TD they suffered from, and their experience. Certain TD items, primarily test debt and code debt, had the biggest impact on stress, though it would be reasonable to assume that other TD-types, which has the potential to cause similar effects, also have an impact on stress, for example, architectural debt. The experience and technical knowledge of the developers seem to have an impact too where TD projected more stress over developers who lack the knowledge on how to deal with the TD while experienced developers could employ effective methods to cope with the TD. In specific circumstances, e.g. at release, there is an immediate, and significant, increase in stress, though this stress is largely caused by the effects of TD, such as bugs, and not by the presence of the TD itself.

4.2 Differences between Developers and Managers

Firstly, both groups seem to get stressed, at least to some degree, from TD but the source of that stress seems to be different. Developers receive most of the stress from TD by the way it interferes with their development and ability to implement or maintain components in a correct and timely fashion, as exemplified by for example interviewee 3: ”I get assigned a new task or I think up a new idea to do but its not possible because the technical debt blocks me and that stresses me, I want to do it” and interviewee 6: ”It’s like, every little change you make, you don’t know if that’s going to make something else break”. The project managers, on the other hand, seem to be more affected from the business perspective of operations, for example, interview 11 when elaborating on the chronic stress of not having automated tests: “When you know that certain parts of the system are less stable because we have not made them completely redundant”, fearing the running operations to fail. Interviewee 5 cites a similar potential situation when running tests: “We are actually running a test of the service where the organization is involved for the first time. So I might be very stressed this afternoon when we go through the results”. The answers, in both cases, were discussed in the context of TD, but instead of describing the debt itself as the developers, they focus on the potential side effects of TD, i.e. increased risk of fault generation [45].

Generally, however, the managers did not seem to get stressed by TD in any meaningful way, at least compared to their other obligations as demonstrated by interviewee 1: “Technical debt is a small percentage of the stress I feel” and interviewee 5: “for my part, I do not think that technical debt is the main stressor, it is other things”. Instead, other stressors take up the majority of the stress, for example, interviewee 1: “What it’s about is that customers’ goals and my long-term goals clash, that’s where I have my stress”. Interviewee 11 was a big exception to this, where TD caused a third of the stress: “So, Technical debt is one third, new establishment project is one third and ongoing operation, making sure everything runs as expected, is one third”.

Investigating the second research question reveals that both groups get stressed by TD, but developers gather their stress from a development perspective, i.e. how much it impacts further development of the product while the managers suffer from chronic stress from the effects of TD, for example lack of redundancy in a system. The project managers did suffer from lower stress levels overall, but it is often the project managers’ decision not to fix the debt and instead to pursue other goals. The managers did mention the relationship with customers and the work with requirements as their main stressors, greatly outweighs the stress caused by TD.

4.3 Attitude towards Technical Debt

Exploring the last theme, attitude, reveals a difference between managers and developers regarding their attitude and work with TD.

4.3.1 Developers on Technical Debt

Developers, when asked about their attitude towards TD for example zero-tolerance or as an effective tool to deliver features in time, were overwhelmingly negative. Most understood that some debt is inevitable and hesitated to call it complete zero-tolerance but would very much prefer to not introduce TD at all, for example, interviewee 2: "Zero-tolerance sound so harsh and wrong. Zero tolerance against, well anything, that's bad right? But yes, we can call it zero-tolerance" and interviewee 7: "I think it would be difficult to say zero-tolerance as then... I think you have to be generally tougher against yourself and others when developing systems to make sure you make the right decisions from the beginning". Other Developers were even more stubborn, preferring complete zero tolerance, for example: "I have zero tolerance against it. I don't want to include it if I see it... I block other developers from including it as well" and "It might sound naive, but I would prefer zero-tolerance".

Some interviewees prioritized certain debt items over others. Documentation debt, for example, was by some seen as unnecessary, or at least not as important: "Other technical debt is perhaps not as important if you understand, for example, documentation, but I would generally want to avoid it" and "Documentation is, in my opinion, only important where you have integration of external components".

4.3.2 Managers on Technical Debt

Project managers, on the contrary, were as a whole more liberal towards the introduction of TD. Interviewee 1, for example, said: "I sometimes encourage my project to introduce technical debt. Simply: I have in some situations told people in my team: 'I am not looking for miss right, I am looking for miss right now'. That is, in fact, an order of technical debt". Interviewee 5, while not quite as liberal and straightforward, follows a more "good-enough" approach: "There is usually a requirement to meet a specified level of test, for example. If we have reached the value that is acceptable, then it is okay". This could be influenced by the managers often not directly receiving the impact of the TD but all managers emphasized the customer's will as their main basis of deciding whether or not to introduce new debt, For example interviewee 1 again: "You must look ahead too. It may be the case that the money is not located where the technical debt is" or interviewee 11: "Sometimes you are forced to take shortcuts to deliver results to a

customer or to avoid delays when other parts of the project a relying on the things we do. In other words, the consequences outside my project or with a customer are greater than if we did not take a shortcut". This indicates that organizations strive towards "good enough" software development where all TD does not need to be addressed, and where a certain level of TD can be accepted as long as the customer is happy with the overall product. This goes against academic results [40, 45] that state that all TD is bad. Thus, indicating a difference in perception towards TD between academia and industry.

4.3.3 Attitude towards Eachother

In general, the developers thought the managers had a decent understanding of their work with TD, especially for critical debt but thought that they did not get the attention they needed to fix the root causes of the issues, for example, interviewee 7 stated: "It depends on which direct consequences the debt has. In the project I am in now you obviously get time to fix debt that impacts the software directly but it's difficult to get time to fix larger things such as architecture" In turn interviewee 4 stated: "The biggest challenge to get rid of this technical debt stuff is because you get pressured to implement so many new things. And it does not matter how much we try to convince managers that don't work with it-development, they never understand how badly it can end".

Managers on the other hand, generally felt that developers understood that introducing some debt was necessary, but pointed out that there often was a group of perfectionists they had to convince: "There is always that group of developers who would rather want to be left alone and refactor their code, there's no end to the fiddling" and "Some people are very zealous and want to do everything perfectly. And that makes me sometimes in my role act as a break pad out of necessity and tell them: 'No, we do not spend time on that now, we must deliver. Does it work?' - 'Yes' - 'Good, then it's done' ". The managers stressed, however, that there was a good dialog between them and that their understanding had grown over time as exemplified by interviewee 1: "I feel that the engineering corps of today have a much better understanding of commercial realities" and interviewee 11: "I think we often have a dialog and when we finally make a decision we know why". The dialog aspect of it was recognized by the developers as well, for example interviewee 11: "Yes, I understand but, on the other hand, it's their job to get products out quickly, or to get us to work faster. But it's our job to have the professional integrity to say: 'This is not good' and ask 'Do you want this software or program to exist for 2 more years or do you want it to work 10 years from now?'. And then you have to find some sort of balance".

To summarize the answers to the third research question, it is clear that TD has a varying impact on the roles involved in development. Developers tend to be more critical about its introduction as they are more directly affected by it. Project managers are, on the other hand, more liberal and prioritize customer satisfaction in the form of "good enough" solutions instead of "perfect solution". This difference in opinion can cause friction in the organization and highlights the need for communication and emphasis on project priorities, e.g. where is TD acceptable within the project and where should effort be spent to minimize TD. This is a non-trivial question that warrants future research, especially taking the concept of "good enough" development into account.

Chapter 5

Discussion

The results from the empirical data suggest that TD, when present, does increase stress. The people who experienced the most stress from TD all cited how the stress interfered with their development as the primary stressor. This effect has been found in other studies, for example, Tom et al. [41], who found that the negative effects of morale were largely linked to how much it interfered with the developer's effectiveness. The main TD types contributing to stress found in this study were test-debt and code-debt. The participants who experienced more stress than the others described their stress as chronic stress constantly in the background and was caused by the presence of the debt itself and how it intefered with their work. The people who did not experience as much stress and described a more acute type of stress often described it in the context of deadlines, meaning that they did not get stressed by the debt itself, but from the deadline. From this data, the study can draw the conclusion that certain debt types, which interferes with the development of a project, seem to be a cause for chronic stress.

The differences between how TD impacts managers compared to developers were not as straightforward. There were some differences in the source of the stress, where developers were more stressed by the actual development whereas the project managers were more stressed from the business perspective and the stability of the product. Some managers did not seem to be affected by the TD at all. It would seem as though the stress and pressure project managers suffer from requirements and customers far exceed the stress they suffer from the presence of TD, given that the TD does not directly threaten the stability of the product. It might even be the case that introducing TD alleviates stress from the project managers as it would allow them to deliver the products in time. It could be assumed that project managers would allocate more time to fix the TD if it was deemed a big enough problem, instead, it seems as though the project managers actively have to request the introduction of TD from the developers.

Lastly, the report investigated if the attitudes between developers and managers

were different. The results suggest that developers are very hesitant to introduce debt into the system. All developers in this study would prefer a strict approach to TD where 6 out of 8 would prefer a zero-tollerance approach. In the literature, Tom et al. [41], found that only developers who were serious about code quality were this negative towards introducing short-term TD. This might suggest that the participants in this study all would be classified into that category. They reported that they sometimes were allocated time for maintenance of the TD but that that time was often not near enough to fix the underlying problems, i.e. the root-causes of TD. The project manager's attitude toward introducing TD into the projects was a lot less strict. This was primarily due to the fact that project managers had to take the customer's wishes and the requirements into consideration. This was expected as the primary role of project managers are to manage the budget and make as much money as possible while simultaneously satisfying the customer. They reported that developers often had an understanding of their position but that they sometimes had to stop certain types of developers from over-refactoring the code. Considering all the above, it is clear that project managers are more liberal towards introducing TD than the developers. This is due to a combination of not being affected by the debt in the same way as developers and budgetary restrictions where they need to keep the customer happy.

5.1 Threats to Validity

Four types of validity threats were recognized during the study, as presented by Runeson et al. [37]. These factors have been presented below, also indicating their severance and means with which they have been mitigated.

Internal validity: Refers to the ability of the research design to answer the research questions. For example, that the interview questions were well formulated to provide sufficient results to answer the research questions. The choice of a case study with interviews is well motivated for this study because of its exploratory nature and the need to elicit answers from industry. The interview guide was also well tested, both through internal review, and through pilot interview. Thus, the raw results are perceived to be of sufficient internal validity. The analysis of the results may be subject of researcher and selection bias as they were only conducted by the author. However, the results were both internally reviewed and provided to the interviewees for further scrutiny. Thus, such biases are unlikely. As such, we perceive the results, although not comprehensive, to be of sufficient internal validity.

External validity: Refers to the generalizability of the results, i.e. how they can be applied in other contexts/domains from the one studied in the study. Here lies the largest threat to the results presented in this thesis since the results were only acquired from one company. However, since this company is a consultancy

company with interviewees having experience from multiple companies and domains, I still claim that the results are of moderate generalizability to the general software development domain. This statement is further supported by related research in the area in other companies and domains, with similar results.

Construct validity: Refers to the selection of subjects and domain to answer the research question. The study was delimited to research within industrial SE practice on how TD affects human factors, in particular stress and attitude towards TD. For this reason, an empirical, industrial study, with interviews, was chosen. According to academic best practice, this is a suitable methodology and method, in particular for exploratory and descriptive studies. The main threat of the study, similar to the threats to the external validity of the results, is that the interviews were only conducted in one company. Thus, limiting the generalizability of the results. However, despite this threat, we still claim that the construct validity of the results to be high, as all presented results are evidence-driven based on interviews with practitioners familiar with the TD concept and its effects on their perceived stress in practice.

Reliability: Refers to the trustworthiness of the methodology and the results. Case studies, per definition, lack reliability since they can not be replicated. The main reason is because the studies themselves affect the phenomenon under study. To mitigate this threat, the research procedure has been explained in as much detail as required to understand the study. A replication package has also been supplied for researchers to scrutinize to see the full chain of evidence leading to the results and conclusions presented in the thesis.

Chapter 6

Conclusions and Future Work

The research on how TD impacts affective states, stress, and morale is a new and upcoming subject with sparse and fragmented research. There had not been any specific research conducted on how TD impacts stress in developers and projects directly. The evidence that existed has only briefly mentioned the concept without further investigations or any noteworthy depth. Previous research has also solely focused on the developers' perspective which might lead to biased analyses and solutions.

This study explored the impact of TD on stress in both project managers and developers as well as compared their perspectives on TD through interviews. Results from the interviews suggest that TD has an impact on stress but that certain items, such as test and code debt has more impact than others. Regardless of debt type, however, the debt that interferes the most with a developer's efficiency and development seems to be the root cause of the stress. It would therefore be reasonable to assume that other debt which also interfered directly with development would have a similar effect to test and code debt.

Project managers are not as impacted by stress from TD compared to developers. In cases where the managers got stressed, it was primarily due to the effects on the stability and redundancy of the system that the TD had caused, and not necessarily the TD itself. Project managers were more liberal when it came to introducing TD compared to developers who were much stricter and would prefer something closer to a zero-tolerance approach. The relationship between the two seems to create a balance where project managers push to deliver new features and requirements while developers push for refactoring and increase the quality of the product.

The results of this study is a great indication that TD has an impact on developer and manager stress. This was a case study from one consultant company in Sweden, however. Further research from other parts of the industry is needed to make better and more generalized conclusions. Hence, studies that use a wider

dataset and including quantitive data in the research could strengthen the results even further.

References

[1] Balaji Akula and James Cusick. Impact of overtime and stress on software quality. In *4th International Symposium on Management, Engineering, and Informatics (MEI 2008), Orlando, Florida, USA*, 2008.

[2] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spínola. Towards an ontology of terms on technical debt. In *2014 Sixth International Workshop on Managing Technical Debt*, pages 1–7. IEEE, 2014.

[3] Aamir Amin, Shuib Basri, Mohd Fadzil Hassan, and Mobashar Rehman. Software engineering occupational stress and knowledge sharing in the context of global software development. In *2011 National Postgraduate Conference*, pages 1–4. IEEE, 2011.

[4] Matthijs Baas, Carsten KW De Dreu, and Bernard A Nijstad. A meta-analysis of 25 years of mood-creativity research: Hedonic tone, activation, or regulatory focus? *Psychological bulletin*, 134(6):779, 2008.

[5] Sarah Beecham, Nathan Baddoo, Tracy Hall, Hugh Robinson, and Helen Sharp. Motivation in software engineering: A systematic literature review. *Information and software technology*, 50(9-10):860–878, 2008.

[6] Terese Besker, Hadi Ghanbari, Antonio Martini, and Jan Bosch. The influence of technical debt on software developer morale. *Journal of Systems and Software*, 167:110586, 2020.

[7] Klara Borowa, Andrzej Zalewski, and Szymon Kijas. The influence of cognitive biases on architectural technical debt. In *2021 IEEE 18th International Conference on Software Architecture (ICSA)*, pages 115–125. IEEE, 2021.

[8] Virginia Braun and Victoria Clarke. Using thematic analysis in psychology. *Qualitative research in psychology*, 3(2):77–101, 2006.

[9] Shirley Cruz, Fabio QB da Silva, and Luiz Fernando Capretz. Forty years of

research on personality in software engineering: A mapping study. *Computers in Human Behavior*, 46:94–113, 2015.

[10] Ward Cunningham. The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4(2):29–30, 1992.

[11] Neil A Ernst, Stephany Bellomo, Ipek Ozkaya, Robert L Nord, and Ian Gorton. Measure it? manage it? ignore it? software practitioners and technical debt. In *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, pages 50–60, 2015.

[12] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[13] Sávio Freire, Nicolli Rios, Boris Gutierrez, Darío Torres, Manoel Mendonça, Clemente Izurieta, Carolyn Seaman, and Rodrigo O Spínola. Surveying software practitioners on technical debt payment practices and reasons for not paying off debt items. In *Proceedings of the Evaluation and Assessment in Software Engineering*, pages 210–219. 2020.

[14] Yuko FUJIGAKI, Takashi ASAKURA, and Takashi HARATANI. Work stress and depressive symptoms among japanese information systems managers. *Industrial health*, 32(4):231–238, 1994.

[15] Tsuneo Furuyama, Yoshio Arai, and Kazuhiko Iio. Analysis of fault generation caused by stress during software development. In *Achieving Quality in Software*, pages 14–28. Springer, 1996.

[16] Hadi Ghanbari, Terese Besker, Antonio Martini, and Jan Bosch. Looking for peace of mind? manage your (technical) debt: An exploratory field study. In *2017 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*, pages 384–393. IEEE, 2017.

[17] Graham R Gibbs. Thematic coding and categorizing. *Analyzing qualitative data*, 703:38–56, 2007.

[18] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. Consequences of unhappiness while developing software. In *2017 IEEE/ACM 2nd International Workshop on Emotion Awareness in Software Engineering (SEmotion)*, pages 42–47. IEEE, 2017.

[19] Daniel Graziotin, Fabian Fagerholm, Xiaofeng Wang, and Pekka Abrahamsson. Unhappy developers: Bad for themselves, bad for process, and bad for software product. In *2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C)*, pages 362–364. IEEE, 2017.

[20] Maria Jimenez-Buedo and Luis M Miller. Why a trade-off? the relationship

between the external and internal validity of experiments. *Theoria. Revista de Teoría, Historia y Fundamentos de la Ciencia*, 25(3):301–321, 2010.

[21] Margaret E Kemeny. The psychobiology of stress. *Current directions in psychological science*, 12(4):124–129, 2003.

[22] Philippe Kruchten, Robert L Nord, and Ipek Ozkaya. Technical debt: From metaphor to theory and practice. *Ieee software*, 29(6):18–21, 2012.

[23] Per Lenberg, Robert Feldt, and Lars Göran Wallgren. Behavioral software engineering: A definition and systematic literature review. *Journal of Systems and software*, 107:15–37, 2015.

[24] Sonja Lyubomirsky, Laura King, and Ed Diener. The benefits of frequent positive affect: Does happiness lead to success? *Psychological bulletin*, 131(6):803, 2005.

[25] Andreas Meier, Martin Kropp, Craig Anslow, and Robert Biddle. Stress in agile software development: practices and outcomes. In *International Conference on Agile Software Development*, pages 259–266. Springer, Cham, 2018.

[26] Sebastian C Müller and Thomas Fritz. Stuck and frustrated or in flow and happy: Sensing developers' emotions and progress. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 688–699. IEEE, 2015.

[27] Jesper Olsson, Erik Risfelt, Terese Besker, Antonio Martini, and Richard Torkar. Measuring affective states from technical debt. *Empirical Software Engineering*, 26(5):1–40, 2021.

[28] Jan-Peter Ostberg, Daniel Graziotin, Stefan Wagner, and Birgit Derntl. A methodology for psycho-biological assessment of stress in software engineering. *PeerJ Computer Science*, 6:e286, 2020.

[29] KS Rajeswari and RN Anantharaman. Development of an instrument to measure stress among software professionals: Factor analytic study. In *Proceedings of the 2003 SIGMIS conference on Computer personnel research: Freedom in Philadelphia–leveraging differences and diversity in the IT workforce*, pages 34–43, 2003.

[30] Robert Ramač, Vladimir Mandić, Nebojša Taušan, Nicolli Rios, Manoel G de Mendonca Neto, Carolyn Seaman, and Rodrigo Oliveira Spínola. Common causes and effects of technical debt in serbian it: Insightd survey replication. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 354–361. IEEE, 2020.

[31] Muhammad Mohsin Raza, Nasir Mehmood Minhas, Hameed Ullah Khan,

and Ikram Asghar. Impact of stress on software engineers knowledge sharing and creativity (a pakistani perspective). *International Journal of Business Intelligent*, 4(2):69–73, 2015.

[32] Nicolli Rios, Rodrigo Oliveira Spínola, Manoel Mendonça, and Carolyn Seaman. The most common causes and effects of technical debt: first results from a global family of industrial surveys. In *Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 1–10, 2018.

[33] Nicolli Rios, Rodrigo Oliveira Spínola, Manoel Mendonça, and Carolyn Seaman. Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, pages 3–12. IEEE, 2019.

[34] Suzanne Rivard, Yannik St-James, and Ann-Frances Cameron. sedgwick2013conveniencesoftware project risk drivers as project manager stressors and coping resources. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2011.

[35] Oliver C Robinson. Sampling in interview-based qualitative research: A theoretical and practical guide. *Qualitative research in psychology*, 11(1):25–41, 2014.

[36] Jennifer Rowley and Frances Slack. Conducting a literature review. *Management research news*, 2004.

[37] Per Runeson and Martin Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, 14(2):131–164, 2009.

[38] Steve Sawyer and Patricia J. Guinan. Software development: Processes and performance. *IBM systems journal*, 37(4):552–569, 1998.

[39] Ruslan Shaydulin and Justin Sybrandt. To agile, or not to agile: A comparison of software development methodologies. *arXiv preprint arXiv:1704.07469*, 2017.

[40] Rodrigo O Spínola, Antonio Vetrò, Nico Zazworka, Carolyn Seaman, and Forrest Shull. Investigating technical debt folklore: Shedding some light on technical debt opinion. In *2013 4th International Workshop on Managing Technical Debt (MTD)*, pages 1–7. IEEE, 2013.

[41] Edith Tom, Aybüke Aurum, and Richard Vidgen. An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516, 2013.

[42] Rita Vine. Google scholar. *Journal of the Medical Library Association*, 94(1):97, 2006.

[43] Madeline Weiss. Effects of work stress and social support on information systems managers. *Mis Quarterly*, pages 29–43, 1983.

[44] Claes Wohlin. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*, pages 1–10, 2014.

[45] Jesse Yli-Huumo, Andrey Maglyas, and Kari Smolander. The sources and approaches to management of technical debt: a case study of two product lines in a middle-size finnish software company. In *International Conference on Product-Focused Software Process Improvement*, pages 93–107. Springer, 2014.

Chapter 7

Appendix

List of Keyowrds
Emperical
Industrial
Technical debt
Code smells
Technial debt items
Non-optimal
Design architecture
Good and bad code practices
Guidelines for software development
Coding standards
Human factors
Soft factors
Human in the loop
Behavioral software engineering

Table 1: List of keywords provided by the University Advisor