



Degree Project in Industrial Management

Second cycle, 30 credits

Technical Debt in Swedish Tech Startups: Uncovering its Emergence, and Management Processes

VILLE ABRAHAMSSON

VICTOR HOLMQVIST

Technical Debt in Swedish Tech Startups: Uncovering its Emergence, and Management Processes

by

Ville Abrahamsson
Victor Holmqvist

Master of Science Thesis TRITA-ITM-EX 2023:271
KTH Industrial Engineering and Management
Industrial Economics and Management
SE-100 44 STOCKHOLM

Technical Debt in Swedish Tech Startups: Uncovering its Emergence, and Management Processes

av

Ville Abrahamsson
Victor Holmqvist

Examensarbete TRITA-ITM-EX 2023:271
KTH Industriell teknik och management
Industriell ekonomi och organisation
SE-100 44 STOCKHOLM



KTH Industrial Engineering
and Management

Master of Science Thesis TRITA-ITM-EX 2023:271

Technical Debt in Swedish Tech Startups: Uncovering its Emergence, and Management Processes

Ville Abrahamsson
Victor Holmqvist

Approved 2023-06-06	Examiner Matti Kaulio	Supervisor Lars Uppvall
	Commissioner CodeScene	Contact person Markus Borg

Abstract

Technical Debt (TD) is a concept referring to technical deficiencies and sub-optimal decisions made in software development that may save time in the short term but lead to long-term obstacles. The concept also implies increased future costs, often growing with interest, caused by slower development rates and the need for refactorings. The high-paced environment of tech start-ups often leads to companies taking shortcuts in their development process, prioritizing iteration speed and time-to-market over long-term scalability. Oftentimes resulting in the accumulation of TD through sub-optimal technology choices, modularity, or architecture. Start-ups also play an important role in the innovation of many industries, hence, their contribution to differing markets and new products is valuable. Poorly managed TD in start-ups may lead to large obstacles when initiating a scale-up phase, leading to a possible development in industries slowing down, and it is therefore important to create the best possible conditions for start-ups to succeed in managing their TD, which is where this study aims to provide aid. A multiple-case study was performed with several tech start-ups in the Stockholm area on how TD has emerged and has been managed through its journey. The findings show that start-up companies are often inclined to deliberately accumulate TD in the early stages, in order to facilitate swift market establishment and proof of concept for their product. Further, the negative consequences of early accumulated TD were found to be limited. However, TD should not be left to grow for too long even in a start-up phase, since the findings show that this often results in large costs. Instead, start-ups should plan for the refactoring of early accumulated TD by expecting a technological pivot. Furthermore, in the continuous management of TD, the results show that team composition, including personality traits and in-house competence, often impacts the success of managing TD more than meticulous planning, motivating the management in the start-ups to more thoroughly consider how they build their teams, and what competencies are present in the company, or needed.

Key-words

Technical debt, Start-ups, Technical debt management



KTH Industriell teknik
och management

Examensarbete TRITA-ITM-EX 2023:271

Technical Debt in Swedish Tech Startups: Uncovering its Emergence, and Management Processes

Ville Abrahamsson
Victor Holmqvist

Godkänt 2023-06-06	Examinator Matti Kaulio	Handledare Lars Uppvall
	Uppdragsgivare CodeScene	Kontaktperson Markus Borg

Sammanfattning

Teknisk Skuld är ett begrepp som syftar på suboptimala beslut som fattas inom mjukvaruutveckling som kan spara tid på kort sikt, men som innebär framtida kostnader, ofta med ränta, genom refaktorering eller långsammare utvecklingstid. Den snabba miljön hos tech start-ups leder ofta till att företag tar genvägar i sin utveckling, vad gäller till exempel teknikval, modularitet eller arkitektur. Start-ups spelar också en viktig roll i innovationen i många branscher, och är därför värdefulla i olika marknader och nya produkter. Dåligt skött teknisk skuld i nystartade företag kan leda till stora hinder när man initierar en uppskalningsfas, vilket leder till att eventuell utveckling i branscher bromsar in, och det är därför viktigt att skapa de bästa möjliga förutsättningarna för start-ups att lyckas hantera sina tekniska skuld, vilket denna studie syftar till att bidra till. I den här artikeln genomfördes en flerfallsstudie med flera nystartade teknikföretag i Stockholmsområdet om hur teknisk skuld har vuxit fram och hanterats genom sin resa. Resultaten visar att teknisk skuld inte anses vara dåligt i mycket tidiga skeden, där ackumuleringen av teknisk skuld ofta skapar marknadsetablering och samlar bevis för vad produkten bör vara för, och tidigt ackumulerad teknisk skuld är oftast lättare att hantera än i senare skeden. Teknisk skuld bör dock inte lämnas att växa för länge även i en uppstartsfas, eftersom resultaten visar att detta ofta leder till alldeles för stora kostnader. Istället bör nystartade företag planera för omstrukturering av tidigt ackumulerad teknisk skuld genom att förvänta sig en teknisk pivot. Vidare, i den kontinuerliga hanteringen av teknisk skuld, visar resultaten att team-sammansättning, inklusive personlighetsdrag och intern kompetens, ofta påverkar framgången med att hantera teknisk skuld mer än noggrann planering, vilket motiverar ledningen i nystartade företag att mer noggrant överväga hur de bygger sina team och vilka kompetenser som finns i företaget, eller vilka som behövs.

Nyckelord

Technical debt, Start-ups, Technical debt management

Contents

1	Introduction	1
1.1	Background	1
1.2	Problem	2
1.3	Purpose	3
1.4	Contributions	3
1.5	Delimitations	4
1.6	Outline	5
2	Literature Review	6
2.1	Technical Debt	6
2.2	Technical Debt Management (TDM)	8
2.3	Agile Software Development	10
2.4	Environment of Software Development Start-Ups	11
2.5	Business Value Impact	13
2.6	Summary	16
3	Research Methodology	18
3.1	Research Design	18
3.2	Research Process	19
3.2.1	Literature Review	20
3.2.2	Multiple-Case Study	22
3.2.3	Interview Theory	23
3.3	Data Gathering	24
3.3.1	Interviews	24
3.4	Data Analysis	26
3.4.1	Qualitative Analysis	26
3.5	Rigour & Ethics	28

4	Context	30
4.1	The start-up ecosystem of Stockholm	30
4.2	Economic impact	32
4.3	Societal impact of start-ups	32
5	Results & Analysis	34
5.1	Vision	34
5.1.1	Visions Impact on Scalability	36
5.1.2	Technical debt tolerance	37
5.2	Process	39
5.2.1	Development Timeline	39
5.2.2	Development Process	41
5.2.3	Technical Debt tracking	42
5.2.4	Refactoring strategies	44
5.3	Competence	45
5.3.1	Skills	45
5.3.2	Personality traits	48
5.3.3	Team Composition and Dependencies	50
5.4	Prototyping	51
5.4.1	Prototyping philosophy	52
5.4.2	Implications of early decisions on future scalability	54
5.5	Pivoting	56
5.5.1	Business Influenced Pivoting	56
5.5.2	Re-writes	58
6	Discussion	61
6.1	Emergence of Technical Debt	61
6.2	Management of Technical Debt	65
6.3	Technical Debt and Scalability	67
7	Conclusion	70
7.1	Theoretical Implications	71
7.2	Practical Implications	73
7.3	Limitations	74
7.4	Future Work	75

References

77

List of Figures

2.5.1 Conceptual model of Prioritization Process (Besker et al. 2019)	15
3.2.1 Research Process	20
3.4.1 Code Tree - All themes Included	28
4.1.1 Swedens Start-up Ecosystem Ranking (StartupBlink 2022)	31
4.2.1 VC Funding Growth 2017-2021, (Institute 2022)	32

List of Tables

1.6.1 Study Outline	5
3.2.1 Keywords used in Systematic Literature Review	21
3.3.1 Interview overview	25
3.3.2 Interviewed companies	25

Acronyms

ASD Agile Software Development

TD Technical Debt

TDM Technical Debt Management

ATD Architectural Technical Debt

Acknowledgements

Without the help of great people, this study would not have been possible. Firstly, we would like to extend sincere gratitude to Lars Uppvall at KTH for his excellent guidance and mentorship, in his role as supervisor for this master thesis. Secondly, Mattias Wiggberg at KTH has provided great support and feedback in his role as a seminar leader, for which we are grateful and appreciative. Thirdly, we would like to thank Markus Borg at Codescene, for his expertise which has contributed greatly to the choice of research area and the topic of this thesis. Finally, we would like to thank our classmates in KTH Industriell Ekonomi I-18 for joining us in persevering through pandemics, and exam periods over the last five years.

Lastly we would like to thank all the participants in the study for their invaluable insights, compelling discussions and great attitudes. The interviews conducted were incredibly rewarding, the persons interviewed were inspiring and extremely competent, and we are more than grateful for the opportunity to take part of your competence and experience.

"I'm convinced that about half of what separates the successful entrepreneurs from the unsuccessful ones is pure perseverance - Steve Jobs"

Chapter 1

Introduction

1.1 Background

Technical debt (TD) is a concept prevalent in the field of software development. Software developed in a hasty manner without proper planning is often harder to understand, harder to maintain and harder to extend with new functionality. Technical debt implies the future cost of rewriting or refactoring existing code, necessary to extend it with new functionality or patch existing defects. In other words, the amount of work needed to rectify shortcuts taken earlier in the development process. Without care and attention, TD is likely to build up in software projects over time, ultimately reducing the value of the technological assets and slowing down the potential rate of development. Short-term gains in development speed are likely to have negative effects on the long-term development speed (Martini et al. 2015). An estimate states that 42% of software developers' time is wasted dealing with TD. Further, code of poor quality is costing corporations \$85 billion annually (Tornhill and Borg 2022).

Most software projects these days employ an agile development process. Agile software development (ASD) has shown several benefits over traditional, waterfall-oriented development processes. Such as faster times to market, higher software quality, and higher customer satisfaction (Baham 2017). These are attractive traits in today's fast-moving markets where competition can be fierce. Fast product iterations and response times to customer requests can yield a competitive advantage. Although there are several benefits to using an agile development process, some research has attributed it as a source of TD; The short timeframes and focus on "the next product feature" can

create a narrow, short-term development focus at the expense of a sustainable long-term one. Short sprints can create incentives for shortcuts and short-term time gains, at the expense of introducing TD that will have to be handled in the future (Rios et al. 2019). At the same time, an agile workflow is also credited as one of the best ways of managing TD in software projects over time. The adoption of agile methods, such as partial refactoring and test-driven development, can over time help reduce and manage TD in software projects (Bomfim and Santos 2017; Codabux and Williams 2013).

Technical debt is not a binary concept tied to any specific market or technology. It is inevitable and all software projects have TD to some extent (Allman 2012). However, some software projects are more prone to accumulating TD than others and the direct effects of TD differ from project to project. One type of software project susceptible to the growth of TD, as well as sensitive to the negative impacts of TD, are start-ups and other projects in a growth phase. These projects typically pivot frequently, to adjust to previously unknown market demands or explore new potential products (Cico et al. 2021b). A start-up is often born out of a single idea that is tested and refined to find the right market fit. In these early days of a project where a large amount of uncertainty exists, the software is often developed without much attention to long-term maintainability since each new product feature is unproven, and there is a large probability that it won't make it to the next product iteration. Under these conditions, large amounts of TD are likely to build up quickly. If the start-up is proven successful and receives users, the development of new product features and improvements of existing ones will be slowed down by the already existing TD (Njima 2019). This issue may hinder many start-ups from reaching their full potential and stops them from contributing to the innovation and development of tech. Hence, since there aren't extensive studies about TD and how to approach it in the start-up scene, increased knowledge surrounding TD for start-ups is needed to facilitate the successful growth and development of new technologies.

1.2 Problem

Technical debt is an inevitable byproduct when developing software. Unless managed, TD is likely to build up over time and cripple the development speed of new product features. However, managing TD is not simply a matter of setting aside the time to do so. By promoting development practices that minimize the emergence of TD, or

prioritizing efforts to reduce the amount of existing TD, time and resources will be diverted from the development of new product features (Besker et al. 2018). From a business value perspective, new and improved product features that create value for customers are what create value for the business (Besker et al. 2018). Managing TD is essentially a problem of balancing short-term and long-term rewards (Njima and Demeyer 2019). This is a pressing issue for start-ups and other projects that frequently pivots.

In the context of developing new technologies and innovation, start-ups play a major role (Publishing n.d.). Start-ups are often the source of radical innovation as well, with examples such as Facebook, Amazon, and Spotify all making huge impacts on a global scale. However, many factors in play determine a start-up's success and ability to grow and contribute to the innovative markets, where TD is present specifically tech start-ups. Technical debt may amount to 20-40% of their entire technology estate (Dalal et al. 2023). Since TD may become such a hindrance to the growth of start-ups, and effectively cripple the evolution of the company (Fatemi 2023), the subject of TD in start-ups must be extensively studied, to prevent start-ups from failing to contribute to the global development because of a failure in managing TD.

1.3 Purpose

The purpose of this study is to explore how technical debt emerges and is managed in start-ups, and how much decision-makers within the start-up value the costs and benefits of accruing TD early. Furthermore, the study aims to explore how TD affects the fast-paced development of a start-up as it transitions into a scale-up phase.

"How does TD emerge in Swedish tech start-ups?"

"How is TD managed and viewed in Swedish tech start-ups?"

"How does the management of TD translate into the scale-up phase, and how do the start-ups ensure that the product is not hindered in its scalability by TD?"

1.4 Contributions

The start-up scene is predominantly not focused on managing its TD both because of lack of funding, as well as deadline urgency (Besker, et al. 2018) and the desire to

rapidly test different product iterations. This may however become costly since many start-ups are very focused on the short-term gains to please investors and get further funding, and the accumulated TD may infer large costs in a scale-up scenario. This study will hopefully show the key factors of an early stage in a company's life, to handily tackle TD and create a smoother scale-up.

1.5 Delimitations

As software start-ups operate in a wide range of industries, it is not possible in the scope of this thesis to examine each industry in that software start-ups are active. Hence, the cases included will be considered a sample of industries, with the effects on start-ups in general as the main point. Furthermore, the thesis is conducted in Stockholm, creating a delimitation in geographical aspects. The scope is narrowed to the Stockholm area, where tech startups are prominent, and the study will reflect this sample. Hence, the results are subject to change if the study would have been replicated in another area.

1.6 Outline

Chapter	Description
Chapter 1 - Introduction	The first chapter serves to introduce the background of the subject, and state certain conditions on which the study operates.
Chapter 2 - Literature Review	The second chapter aims to establish and summarize what the relevant existing literature is on the subject at hand. Furthermore, this chapter aids in steering the purpose and research questions of the study to be able to contribute new insights into the area.
Chapter 3 - Methodology	The methodology chapter describes the research design and the methods used in the research process, including data gathering and analysis. Ethics, validity, and quality are also discussed.
Chapter 4 - Swedish Start-up Scene	The Context chapter aims to set the scene for the subject area and describe start-ups' contribution to the development of the Swedish market.
Chapter 5 - Results & Analysis	The chapter presents the results from the semi-structured interviews from the multiple cases studied. The results are presented along with the analysis conducted, with permeating themes through the cases.
Chapter 6 - Discussion	The chapter discusses the results and their implications on existing literature, by comparing the gathered knowledge from the literature review with the empirical findings.
Chapter 7 - Conclusion	The chapter summarizes the study and presents key conclusions from the results and analysis. Recommendations for future work on the subject are also included.

Table 1.6.1: Study Outline

Chapter 2

Literature Review

2.1 Technical Debt

The term Technical Debt (TD) was first coined by Ward Cunningham in 1992, drawing comparisons between sub-optimal software implementations with monetary debt that grows over time with interest (Cunningham 1992). Technical debt in software emerges from developers taking shortcuts, opting for the easy or quick solution to a problem, that yields short-term rewards in terms of time and cost savings but might have negative long-term effects (Alves et al. 2016). Ward Cunningham's initial definition of TD mainly referred to immature, "first time code" (Cunningham 1992). However, the term has over time been expanded to include a wide range of technical deficiencies. A mapping study by N. Alves et al. (2016) found 15 types of technical deficiencies referred to as TD in published articles (Alves et al. 2016), including:

- **Design debt:** Violations of good object-oriented design principles. For instance, too large or tightly coupled classes.
- **Documentation debt:** Missing or inadequate documentation.
- **People debt:** People issues in software organizations that can delay or hinder development activities. For instance, expertise is concentrated on too few people.
- **Test debt:** Inadequate testing activities or test coverage.
- **Architecture debt:** Problems related to product architecture. For instance low degree of modularity.

Typically, the longer TD is left unattended, the more expensive it will be to rectify (Holvitie et al. 2018). TD reduces the maintainability of code and slows down the development rate of new functionality (Cico et al. 2021b). Taking on TD can be compared to taking a loan today against the time required to do future work (Rubin 2012). A study found that 23% of developers' time is wasted on handling existing TD (Besker et al. 2020). Further, TD often compounds, aggravating the negative consequences in a non-linear fashion (Rubin 2012). When the amount of TD reaches critical levels, "development crises" can occur, meaning that all development on a project stops, and is unable to continue until a major refactoring has taken place (Martini et al. 2015).

Just as TD can manifest itself in multiple ways, it is also incurred for different reasons. The following taxonomy introduced by Steve McConnell categorizes different types of TD (Kruchten 2012):

1. *Debt incurred unintentionally due to low-quality work*
2. *Debt incurred intentionally*
 - (a) *Short-term debt, usually incurred reactively, for tactical reasons*
 - i. *Individually identifiable shortcuts (like a car loan)*
 - ii. *Numerous tiny shortcuts (like credit card debt)*
 - (b) *Long-term debt, usually incurred proactively, for strategic reasons*

As outlined in McConnell's taxonomy, different types of TD are incurred for different reasons. Type 1 TD is typically a by-product of unintentional developer mistakes or low-quality work, often referred to as "code smells". Type 2 TD is often introduced at a higher, architectural level than type 1 TD (Kruchten 2012). In addition, type 2 TD is the result of deliberate actions and decisions made. TD is in other words not always the result of oversights, mistakes, or incompetence, it can be incurred intentionally to reap the short-term rewards of moving quickly (Yli-Huumo et al. 2014). Such as capturing market share or collecting early customer feedback (Yli-Huumo et al. 2014).

Architectural technical debt (ATD) can be considered TD caused by the fundamental design decisions of a system, for instance, the choices of structure, frameworks, technologies, and programming languages (Verdecchia et al. 2021). While being sound decisions at the time of initial implementation, they can significantly hinder

progress in the future (Verdecchia et al. 2021). According to Soliman et al. (2021), architectural design decisions have the biggest impact on the quality of a software system. Architectural decisions are hard to change after their implementation and can lead to TD if future changes to the systems are costly or limited (Soliman et al. 2021). An optimal system architecture can be considered the trade-off between technological considerations and the needs of the business (Martini et al. 2018b). However, the balance of such a trade-off might change over time as business requirements change. Parts of the system might not prove modular enough (Martini et al. 2018b). Anticipating a solution that will prove optimal for years is a difficult task, which calls for the need to monitor costly ATD over time (Martini et al. 2018b). However, identifying, tracking, and ultimately removing TD at the architectural level is much more difficult and much more expensive than TD on the code level (Verdecchia et al. 2021).

2.2 Technical Debt Management (TDM)

TD is not a consequence that only affects some products or systems. It is inevitable and the question is not how to eliminate it, but rather how to manage it (Oliveira et al. 2015). The role of TDM is to control, prevent, track, and reduce TD, and can be achieved with the help of processes, techniques, and tools (Yli-Huumo et al. 2016). However, implementing TDM is not a trivial task (Yli-Huumo et al. 2016). Identifying, quantifying, visualizing, and tracking TD are all challenging activities (Power 2013). Multiple aspects of organizations and development processes complicate the effective management of TD. Oliviera et al. (2015) presented three common examples of such friction:

1. The question of responsibility - Is it the development team, the product owner, or the scrum master that is responsible for managing TD?
2. The question of incentives and perceived rewards - Do the individuals responsible for planning and product decisions understand the implications of TD? If not, the reduction of TD is often discarded in favor of product feature development.
3. There are rarely pronounced goals regarding TD and perceived problems of TD are rarely documented or structured.

A means of elevating the problems of managing TD is through visualization. By visualizing TD, the development team, product owners and business people can

have a shared context, facilitating meaningful conversations in a concrete way that everyone can understand (Rubin 2012). This can be achieved in multiple ways. For instance through the use of defect tracking software such as static code analyzers, by documenting TD artifacts in the product backlog, or by documenting TD on its own backlog (Rubin 2012).

As previously mentioned, visualization, along with identifying and tracking TD, can be achieved with the help of code analysis software (Li et al. 2015). Such tools can help indicate the existence of TD, by identifying areas of problematic source code, for instance, code smells, low test coverage, code duplication, or highly nested logic (Li et al. 2015). However, these tools are not applicable to all kinds of TD, as the availability of tools that can automatically detect architectural TD is lacking for instance (Verdecchia et al. 2021).

Refactoring refers to activities aimed at improving the internal structure of software, in a way that does not affect the external behavior (Becker et al. 1999). In the metaphor of sub-optimal software decisions being financial debt, refactoring can be considered a tool for repaying debt (Wiese et al. 2021). Refactoring is widely regarded as an effective way of reducing the amount of TD in software systems and is considered a central TDM activity (Holvitie et al. 2018). However, refactorings are often overlooked in prioritization and are often carried out once a development crisis is imminent, in a reactive fashion (Martini et al. 2015). In a multiple case study conducted at 5 large Scandinavian companies, multiple different refactoring strategies at the architectural level were compared (Martini et al. 2015):

1. *No refactoring* - Refactorings were only carried out at the point of a development crisis, which in turn led to development crises occurring more often, over time hindering responsiveness and affecting long-term productivity. Also, the short-term time gains of not spending time on refactorings were partially negated by higher levels of ATD.
2. *Complete refactoring* - A strategy to maximize the amount of refactoring, attempting to completely eliminate the occurrence of TD. This was proven impossible because unknown TD was constantly being accumulated.
3. *Partial refactoring* - The actual outcome of *complete refactoring*, where short-term and long-term TD was refactored regularly.

Partial refactoring was proven the most successful long-term refactoring strategy, pushing the crisis point to a later point in time compared to that of *no refactoring*, while being able to benefit from faster development speed (Martini et al. 2015).

Since Ward Cunningham first defined TD in 1992, much research has been done on the subject, and understanding of the concept has advanced in this field of research (Fungprasertkul and Bahsoon 2021). However, a knowledge gap between TD concepts and their practical usage remains (Fungprasertkul and Bahsoon 2021). In a study conducted among software development practitioners, 79% of respondents answered that the lack of awareness negatively impacted the amount of TD in software projects (Ernst et al. 2015). Further, 65% of respondents had no clearly defined TDM practices, and 42% of respondents stated that business managers and executives were largely unaware of the companies' TD (Ernst et al. 2015).

2.3 Agile Software Development

Agile software development refers to lean, iterative, and lightweight development practices (Baham 2017). A key benefit of ASD over traditional waterfall models is the ability to quickly release new software functionality to customers (Codabux and Williams 2013). Another benefit of ASD is adaptability to change (Allman 2012). As systems are developed and tested, requirements almost always change as the company and its customers gain experience with the system, meaning that requirements and system specifications rarely can be finalized before development begins (Allman 2012). This often leads to agile development methods being preferred over sequential project models when business needs and technologies change rapidly (Codabux and Williams 2013). Agile software development (ASD) has achieved widespread adoption in the industry and is currently the dominating development method (Rios et al. 2019). However, as ASD mainly focuses on functionality and short project cycles, other concerns such as good design, test coverage, and other long-term considerations might be neglected (Codabux and Williams 2013). Although fast iterations enabled by agile work methods could facilitate timely reduction of TD, the opposite often occurs (Oliveira et al. 2015).

2.4 Environment of Software Development Start-Ups

TD challenges in many forms are prominent in software start-ups (Besker et al. 2018). The challenges that are prominent in software start-ups with regards to TD were not studied to a larger extent before 2013-2016 (Cico et al. 2021a) where the need for these studies in context was reported. These challenges occur in different life-cycle phases because of the urgent need to meet demands set by investors, or the urgency to reach a market or add a feature to claim market positions. These occasions need compromises which in turn accumulate TD. It is becoming even more urgent for software start-ups that are transitioning from an early phase to a growth phase (Besker et al. 2018). Furthermore, the perception of TD in start-up phases is considered minimal in regards to the other stages of the business lifecycle (Cico et al. 2021a).

In a study by Cico et al. (2021), they stated that start-ups at an early phase are more prone to accept or ignore TD and that the perception is that accruing TD at an early stage is not necessarily bad. Cico et al. (2021) state as one of their key findings that accepting TD at an early stage, and accumulating it by developing prototypes for example to establish the market validity and make sure that the product requirements are in place. Furthermore, they argue that in the same way that it may be beneficial to accumulate TD at an early stage, it is important to manage, prevent and avoid TD in a growth phase. This since accumulated TD may lead to a software development crisis which has the potential of slowing down, or completely halting the product, leading to major maintainability issues or completely scrapping the product (Cico et al. 2021a).

With start-ups, there are trade-offs when deciding whether to handle TD or add new features to satisfy customer demands (Njima 2019). It is therefore important to find ways to find viable trade-offs at early stages (Njima 2019). Furthermore, achieving these trade-offs, and consequently optimizing the business is vital to achieve the results wanted (Njima 2019).

As the companies evolve through inception, stabilization, and growth, the management of TD throughout the phases has benefits to both increased productivity, as well as time-to-market (Njima and Demeyer 2019). A miscalculation in the engineering of the product could lead to the accumulation of TD, and an under or over-engineering

of the product, contributes to wasted resources and missed market opportunities (Njima and Demeyer 2019). These trade-offs, as previously mentioned, may be aided through certain development practices. One process that is most commonly used is agile as presented earlier, however, more linear and standardized practices may also add benefits (Njima and Demeyer 2019). These more linear engineering processes are not commonly used in start-ups, as the perception of these is that they are more accustomed towards larger companies and not flexible enough for the start-up environment, along with a negative view of them being bureaucratic and costly (Njima and Demeyer 2019). However these types of standardized processes may aid in time to market and the economy of production, which both are critical for start-ups (Njima and Demeyer 2019). Furthermore, the lack of any processes, often resulting in ad-hoc responses to market movements or customer feedback, is also a source of TD in start-ups (Njima and Demeyer 2019)

In addition to the processes by which software start-ups operate, and the accumulation of TD in regards to that aspect, there are other more soft aspects of the start-ups that have a major effect on the TD. The competence within start-ups, or the lack thereof, affects the TD in the companies (Njima and Demeyer 2019). The skill within a company affects the code quality, where poorly written code often doesn't make it into the final product, leading to a waste of resources, or generating TD later down the line (Njima and Demeyer 2019). Furthermore, the stability of the tech teams is also influential on TD within start-ups. For teams that have varying consultants, altering colleagues, and interns, and varying team sizes, the TD amplifies. This is often due to handover processes not being followed, limited competence and experience, or not enough documentation available for the will resume the work (Njima and Demeyer 2019).

Furthermore, the concept of pivoting is common for start-ups (Besker et al. 2018). Pivoting is a phenomenon that has a high presence among software start-ups which is where a company has a change of course or a strategic alteration in the business idea which start-ups use to grow or survive the unforgiving environment (Besker et al. 2018). start-ups are facing challenges when dealing with TD, (Apa et al. 2020) as well as handling pivoting in a scale-up phase (Nguyen-Duc et al. 2015). For start-ups that are in a phase where pivoting is prevalent, managing and avoiding TD significantly reduces the likelihood of technology pivoting (Besker et al. 2018). However, TD does not have a major influence on business-related pivoting decisions. (Besker et al.

2018)

There are a few studies relating to empirical evidence related to TD in a start-up environment (Besker et al. 2018). There has however been a study that presents the need for models, frameworks, methods, and tools that aids in managing TD (Martini et al. 2018a). Moreover, in a study from 2021, Cico et al. argue that for start-ups transitioning to a growth phase, there are four dimensions to TD. These four are managing, accepting, avoiding, and ignoring technical debt (Cico et al. 2021a).

2.5 Business Value Impact

As previously stated, TD is a form of short-term gain, to for example satisfy urgent customer needs, motivation for investors to further the funding, or to meet important deadlines, at the cost of paying this shortcut back at a later date. For fast-moving software development companies, TD may stack up quickly if not handled regularly. Industries operating in "high clock-speed" environments with high competitive threats, such as tech start-ups, are more prone to accumulate TD. (Banker et al. 2021). Furthermore, companies that accrue TD within key areas such as their CRM system impact the revenue for the concerned company (Banker et al. 2021). Not identifying or ignoring TD may then in turn fault the business plan presented for investors (Kruchten 2012), which will have a negative effect on funding and business value in the long term.

For businesses that have not begun to manage TD in their project routine, the prospect of tackling all the accumulated TD may seem daunting. The initial cost of starting to track and manage TD (Guo et al. 2016) is also a hindrance for many smaller companies where funding plays a large role. However, actualizing the initial cost at an early stage may create larger long-term value as recurring costs of handling TD, when being made routine, quickly falls down to reasonable levels (Guo et al. 2016). An early investment in TDM may therefore ease the larger initial cost of the compounded TD at a later stage, as well as create a project environment where business plans and future deadlines are followed to a larger extent. In addition, strategically managed TD can help companies take advantage of time-sensitive opportunities and fulfill market needs (Kruchten 2012).

In the current environment, there is no uniform way of prioritizing TD (Besker et al.

2019). Prioritization processes vary extensively between businesses and organizations, leading to equally varying approaches to handling TD. The refactoring of TD is not always dependent on a reactive or proactive strategy but is often influenced by the pressure put on the team by internal or external customers. Besker et al. (2018) identify that the refactoring of TD is often down-prioritized by the teams because of pressure to deliver customer value and meet delivery deadlines. The pressure of external customers is often larger since this has a direct effect on visible company performance, such as the implementation of new features. It is also evident that the prioritization of these TD items is not decided by any standardized approach, but instead is in large part decided by the gut feelings of the participants. The prioritization process is described to be largely dependent on how the discussions in the meetings develop, as well as which employees are present. These gut-feeling decisions aren't however void of any type of firm-based strategies, since most gut-feeling decisions consider aspects such as *Quality*, *Financial* and *Product & Business Needs*. Furthermore, the decisions made on TD were mostly reactive, i.e. handled when they presented a problem, whereas the decisions were seldom proactive where they took actions to create long-term value. (Besker et al. 2019)

Besker, et al. (2019) identified that decision-makers not only consider the impact on the business when determining how they prioritize the TD but also several other aspects. Besker, et al. (2019) produced a conceptual model as seen in *Figure 2.1.1*, where they visualize how companies approach their TD management. This figure illustrates that certain impacts on the business are mostly considered as an approach to TD refactoring, but the actual decisions consider multiple extended factors. The model brings into light many nuances of how decisions are made in a dynamic project environment, which is important to consider when conducting further research within the TD sphere.

The model was developed by Besker, Martini, and Bosch by performing a literature review on the prioritization process of TD. In that respect, this conceptual model summarizes previous studies in a structured and adaptable manner that leave us with a strong opportunity to develop this knowledge and utilize it to analyze aspects of TD that concern decision-making in project environments.

From the IEEE International Conference on Software Maintenance and Evolution, de Almeida et al. (2018) state that a perspective on TD that is supported by business values

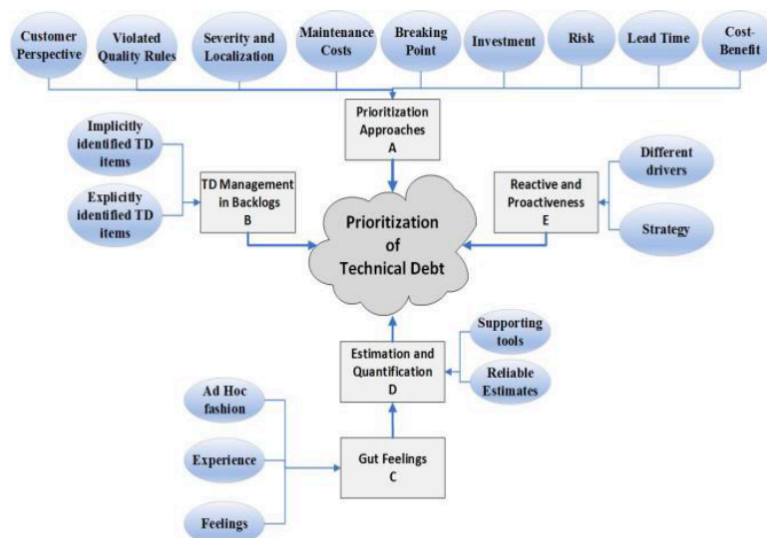


Figure 2.5.1: Conceptual model of Prioritization Process (Besker et al. 2019)

and priorities is needed. As the report from the Dagstuhl Seminar 16162 (Avgeriou et al. 2016) argues, where 33 practitioners from both academia and industry participated, considering business value is critical when creating TDM practices. It is also stated in their road map for the research of TD, that being able to demonstrate the benefits of including TD issues in managerial decisions is important for future research. The tension that exists between software development and business priorities is prone to create TD items (Almeida et al. 2018). Oftentimes, management and business executives are more keen to consider the business value of a software project, where this communication down to the software developers may be lacking, and vice versa.

From the results of the multiple-case study performed by de Almeida et al. (2018), an account manager stated that "sometimes the level of detail from one side, and lack of proper understanding from the other can influence the decision about how urgent or critical a TD item is." Sometimes the technical team discusses and presents the code, and problems concerning it, at a too technical level, that the business management stakeholders cannot understand the issue (Almeida et al. 2018). This may lead to the technical team adding complex explanations to make it seem like the problem is more impactful and urgent than it might be from a business perspective. However, in the same way, business management and business analyst may use arguments that are difficult for the technical team to understand, to motivate certain tight deadlines or inclusion of added features and down-prioritize resolving other technical issues. The

knowledge gaps and communication flaws between the division are prone to develop TD, which may in turn have a impact on the immediate business value. Following this, de Almeida et al. (2018) propose that a business process management approach should be taken in conjunction with the technical impact of TDM. The business process management approach should include tools to identify business aspects when handling a TD item. With examples such as the measurement of urgency, financial aspects, and criticality of the items.

2.6 Summary

The literature has examined the existing body of knowledge regarding TD, how it can be handled, and how it affects start-up companies.

Technical debt (TD) refers to problematic or sub-optimal software artifacts that create friction in maintaining and extending the software. Technical debt is generally something companies want to avoid since it consumes large amounts of developer resources and hinders responsiveness to market opportunities. However, it can also be taken on intentionally for short-term time gains. Technical debt can occur for different reasons, ranging from poorly written code to knowledge dependencies concentrated in a small number of individuals.

Managing TD is no trivial task. It is a somewhat abstract concept that is hard to convey to non-technical people in a meaningful way. TD can be tracked and made visible through project management practices and specialized software. It can be reduced through activities such as refactoring, where code and architecture are rewritten or restructured in a way that improves its modularity or performance, without changing its external behavior. There are different strategies as to when refactorings should be performed, each with its strengths and weaknesses. However, refactoring deficiencies at regular intervals is usually the best option for long-term sustainability. Agile development practices are often attributed as favorable for software development and can facilitate effective TDM. However, it can also lead to more TD if not performed with long-term maintainability in mind. In practice, many companies lack a TDM perspective in their daily operations, potentially posing an issue preventing TD from accumulating in software projects.

In terms of how the business impact affects the accumulated TD, studies show that

the prioritization process often leads companies to consider many factors that may show immediate value, before initiating the to manage their TD. The cost of postponing the initiation to manage, or handle a particular issue, of TD will however lead to exponentially increasing costs. However, if the companies are to start managing their TD at an early stage, the costs may initially prove high, but will soon come down to reasonable levels and will lower the overall costs of their TD over time. A key takeaway is that it is critical to consider business value and effects when creating practices for managing TD, and in their approach to TD overall.

Technical debt can pose a risk for any software company. However, start-ups are in some cases more prone to accumulating TD, and more sensitive to its negative consequences. TD is likely to build up in start-ups as they are affected by an urgency to reach the market, add features and meet expectations set by investors. Problems for start-ups can arise when transitioning from an early phase to a growth phase if large amounts of TD might have been built up while prototyping and establishing market validity. Moreover, start-ups commonly pivot, where companies change course regarding strategy or business plans, to survive. start-ups also operate under certain conditions that can affect the amount of TD accumulated. For instance limited competencies of the founding team as well as stability of the tech team, where a varying number of consultants and interns might be working on the product. The number of studies conducted on TD in the context of start-ups is limited, calling for more research investigating the mechanisms affecting start-ups' attitudes toward accumulating and managing TD as companies transition from a start-up to a scaleup phase.

Chapter 3

Research Methodology

3.1 Research Design

The research design was constructed by adhering to the several layers of the Research Onion (Saunders 2015). The first choice to be made was that the study would be of the exploratory characteristic. This is suited to the nature of the research question since the subject of TD is not extensively studied in the start-up area, and studies on the implementation and consequences of TD of an agile development approach are lacking. Exploratory research is beneficial since it is flexible and adaptable to change (Saunders 2015), which is applicable to this study since the nature of the research question gives an environment that is prone to change when new data is gathered and analyzed.

In an exploratory analysis such as the nature of the conducted study will be, it is important to not avoid being sensitized by existing ideas when analyzing the data gathered (Saunders 2015), or when performing the data gathering. Since the question has not been extensively studied, the aim is to form a more informed view of the current situation. For this study, the data gathered from interviews may be subjective due to differing experiences, priorities, or knowledge since the process is to perform a multiple-case study with several different software start-ups. An exploratory study also enables the interview questions to be openly formulated, allowing for discussion and in-depth reflection (Saunders 2015). Seeing as the managerial issues revolving around TD is both complex and may be subjective to individuals' own experiences, there is a need to be reflexive and holistic in the analysis. This infers an abductive approach, where the final research question and hypotheses were set by a systematic literature

review, where the hypotheses and aim were adjusted based on the data gathered.

To answer the research question, this study will conduct an exploratory qualitative multiple case study. Case studies are preferred when handling "how" research questions, and the nature of the implied study (Yin 2009). The case study method is beneficial since it allows the authors to retain a holistic view of the subject, and analyze meaningful characteristics such as managerial and organizational processes, and how companies mature during their life cycles (Yin 2009). This leads it to be the most suitable approach for the study since the study aims to relate the management and emergence of TDM to project management, organizational values, how start-ups transition from an early phase to a scale-up phase, and how the management of TDM evolves. A multiple-case study was chosen since the environment and industry of tech start-ups are so different, and using a single case study would not have any explanatory attributes if not validated. Hence, a research design that included multiple cases to further validate and bring robustness to the study was chosen (Yin 2009). By studying multiple organizations and business ventures individually, the aim is to generate generalizable insights about different phenomena. By analyzing different start-ups in different stages of their journey, insights regarding barriers and success factors of effectively managing td in that specific setting can be acquired. Events and conditions can be analyzed from the context of specific situations, and also from indications across multiple situations.

For an exploratory study, a suitable approach is to use qualitative analysis (Saunders, et al, 2015). The qualitative analysis is in this thesis the main component of the study, where the data from the qualitative analysis will be the foundation of the result for the research question. The subjects for the interview are hence of utmost importance and will be chosen carefully (Baxter and Jack 2008) (Yin 2009).

3.2 Research Process

The study was conducted as an iterative process, with all components of the study being revisited during the course of the study, as new data was continuously gathered and reviewed. Due to the nature of the abductive approach, and the fact that it is exploratory, the research process and aim of the study were given the flexibility to adapt to the newly gathered empirics (Saunders 2015). The process was divided into four parts with, a pre-study as the first phase of the study.

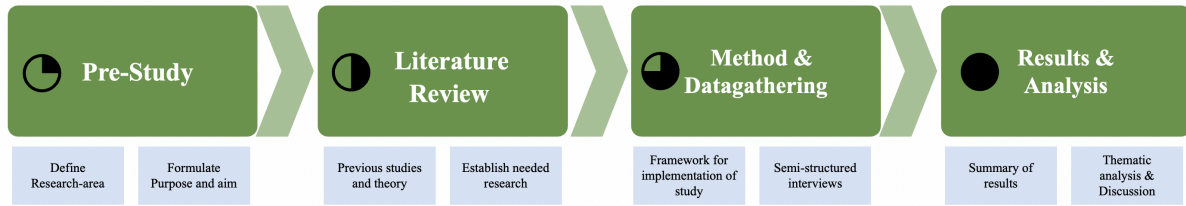


Figure 3.2.1: Research Process

The pre-study consisted of defining the research area of interest in collaboration with the supervisor CodeScene. CodeScene is a company that has developed a tool for identifying td within your company, both on a technical level in the code, and also in the organization. The interest in the study came from the fact that CodeScene believes that there is a need for further research surrounding td in almost every area. Hence, the requisites of a specific project were not needed, and the authors had free reign over what the subject of the study would be, with the supervision of CodeScene in addition to the supervisors at KTH. The research area was then decided via an initial literature review, to understand how the study could contribute to the existing literature (Creswell 2009). The field of study was also influenced by the author's previous experiences, education, and interest. Furthermore, the decision to examine the start-up scene, in particular, was influenced by the low amount of studies associated with software development companies in a start-up phase, as well as the aim to increase awareness with start-ups surrounding their td, possibly opening up a wider clientele for CodeScene.

3.2.1 Literature Review

A vital part to complete this study was to conduct a thorough systematic literature review. The literature review fulfilled a plethora of important components for several phases of the thesis, including the pre-study. The review aided in establishing the research question. The authors set out with the aim to contribute to the field of TD, and through the systematic literature review, the authors could spot a gap in the literature that hadn't been handled extensively. Furthermore, it aided in creating a theoretical foundation based on the previously conducted research, as well as contributing to how the subject was analyzed.

To conduct the systematic literature review, the tool *Zotero* was used to effectively collect, filter and divide the literature found relevant. In addition to that, the primary

articles were gathered via *Web of Science: Core Collection* which in conjunction with Zotero created an adaptable and relevant foundation for the literature review. The final search was made with the keywords presented in Table 3.2.1.

Title	Topic	Operator
Technical Debt	Technical Debt	AND
-	Management	OR
-	Strateg*	OR
-	Project Management	OR
-	Business Value	OR
-	Business Value Impact	OR

Table 3.2.1: Keywords used in Systematic Literature Review

By using Web of Science, only articles, journals, and seminar reports from acknowledged publishers were included in the results of the search. Hence, the published work has been peer-reviewed and accepted by multiple sources, making the literature found through Web of Science trustworthy and valid. Furthermore, the articles gathered were filtered to be from 2016 and onwards, seeing as there was an increase in theory developed surrounding td in start-ups by this time (Cico et al. 2021a). However, while snowballing, and using relevant articles found in the articles created by the search, literature that was published before 2016 was included when they brought value to the study. During the gathering of the existing literature, the authors tried to observe the existing body without bias and attempted to gather sources proposing multiple points of view on a subject, with several sources confirming the evidence.

The search generated 189 results, which after a first session of filtering by reading through each abstract, diminished to 80 relevant articles. These 80 articles were then thematically coded and labeled with the core themes *Agile Software Development*, *Architectural Technical Debt*, *Business Value Impact*, *Decision-making*, *Identification*, *start-ups*, *Stakeholder Perspective*, *Project Management* and *Success factors and barriers*. This was completed during the pre-study, and by observing what we had learned so far, the research question was formulated. When

the research question had been established, the authors narrowed the search down to the perceived key areas when handling the proposed research question. These themes are handled as subsections in Chapter 2, Literature Review.

For all articles reviewed, for each specific topic, the common denominator that needed to be included was that each topic studied was in relation to td. Furthermore, each article that was included needed to contribute to the in-depth knowledge of the study and aid in the understanding of the subject.

3.2.2 Multiple-Case Study

The reasoning behind the choice of a multiple case study is that the overall study is considered more robust, as opposed to a single case study (Yin 2009). In addition to that, the cases included in the start-up scene all differ in product, vision, work processes, and environment, a single case study will not be able to answer the research question in a robust way. By viewing the different cases as "multiple experiments", and using replication logic to apply the study to the different companies, the findings will be either replicated or disputed within the design of the study (Yin 2009). This will add to the validity and rigor of the study (Shah and Corley 2006).

During the study, and after the first few interviews, thematic analysis was used to identify a theoretical framework with recurring themes throughout the interviews. This is an important step in the replicating process, to establish when a condition is likely to be found, and vice versa (Yin 2009). The framework was continuously developed and finalized as the study progressed, where the relationship between the data and the themes identified was solidified during the process (Baxter and Jack 2008). The certainty that one needs to validate the results to the desired extent, decides how many cases one needs in the study. For the case of the scope of this study, and the extent to which this exploratory thesis indicates certain results, 6-7 cases were considered to be a sufficient amount of validating cases (Yin 2009).

start-ups are often limited by a restricted amount of time and are therefore seldom available for interviews with students. However, the approached start-ups showed a large interest in the subject, and several relevant players decided to participate in the study. The companies that were chosen for the study all contributed with unique data, operating with unique products in differing markets. In this way, with independent data sources in different markets, the results were able to be triangulated in a suitable

manner (Yin 2009).

3.2.3 Interview Theory

In a qualitative study, as well as in research methods in general, the most critical part of the study is a rigorous and thorough data collection (Kallio et al. 2016). The data collection will influence both the quality and trustworthiness (Kitto et al. 2008) of the outcome of the thesis, and impact the quality of the study. Therefore it is crucial that the method for the data collection is well-motivated and suitable to fulfill the requirements of the study.

A popular form of data collection for qualitative analysis is semi-structured interviews. One of the reasons this type of data collection is popular is because of its flexibility and versatility. (Kallio et al. 2016). How rigid the structure of the interview will be is also adaptable, depending on what results are needed, and what the purpose of the study is (Kelly 2010), which in turn adds to the popularity of the method. This method was chosen to be suitable for the current study. This is because, as stated in section 3.1, there is a need for reflection and adaptability to be able to appropriately handle an exploratory study.

Kallio et al, (2016) created in their study *Developing a Framework for a Qualitative Semi-Structured Interview Guide*, a five-phase guide on how to conduct semi-structured interview data collection in qualitative studies. The first phase of conducting a semi-structured interview is to establish the prerequisites of the interview. Several issues are needed to be taken into consideration, where a core question is the depth of the information collected (Kallio et al. 2016). The interviewers need to have previous knowledge of the phenomena, which has been fulfilled by previous exposure to the subject and a relevant line of study. The second phase consisted of retrieving and using previous knowledge. The reasoning behind this phase was to establish a solid foundation of knowledge and a sufficient understanding of the subject. This was completed via the systematic literature review, which through rigorous research was extracted and summarized in section 2.

Phase 3 of the guide was to formulate the semi-structured interview map that was to be used during the data collection. Using the previously collected knowledge, a list of questions that are intended to guide the interview through the topic, and gain relevant responses through the interviews will be formed. The quality and relevance of this list

of questions will heavily influence the quality and impact of the thesis. Therefore, it is vital to be diligent and thorough in producing appropriate questions.

The fourth and fifth phases of the guide include testing the proposed structure to confirm the coverage and relevance of the interviews and presenting the final version. This aided in adjusting and reformulating the questions and structure to better suit the purpose. The presentation of the complete interview structure is presented in the appendix. For further research, as well as duplicating the research to confirm its validity it is important to apply a transparent manner throughout the study.

For an exploratory study where there is limited previous research, a less structured interview is appropriate (Kelly 2010). However, when analyzing large quantities of interviews, it is far easier to analyze a more structured interview. Following this, to appropriately analyze the data, a smaller number of interviews, with higher quality and flexibility was conducted. Social skills in the form of active listening, reflecting, and attention also influence the outcome of the interview. It is therefore important to "be present", and take an active part in the duration of the interview. In addition to the set structure of the interview, a vital part of conducting a semi-structured interview is probing and follow-up questions (Kelly 2010). They are important to fully complete the picture wanted of the discussed topics from the respondent. Less structured interviews such as the ones in the study, play a larger role in extracting the most value from the interviews.

3.3 Data Gathering

3.3.1 Interviews

As previously mentioned, semi-structured interviews were conducted as means of data collection. The interviews were held with technical leads, founders, and experienced developers with technical insight from 7 start-ups. A total of 3 technical leads, 3 experienced developers, and 4 CxOs were interviewed. The choices of interview candidates were based on gathering a holistic view of how td affects each component of start-ups. For developers, it is often important to have a clean back-end, whereas, for C-level executives, it is often more focused on delivering the product at a high velocity with plenty of features to gain the highest customer satisfaction and possible revenue.

Alias	Company	Role	Interview duration	Date
I-01	A	CTO	52min	2023-03-20
I-02	B	CEO	57min	2023-03-21
I-03	C	Tech Lead	52min	2023-03-22
I-04	D	Product Manager	1h02min	2023-04-12
I-05	E	Developer	59min	2023-04-19
I-06	F	Developer	52min	2023-04-19
I-07	D	Lead Architect	1h15min	2023-05-03
I-08	G	Developer	26min	2023-05-04
I-09	B	CPO	42min	2023-05-10
I-010	H	CEO	35min	2023-05-12

Table 3.3.1: Interview overview

All companies interviewed are based in Stockholm, Sweden, ranging from early-stage start-ups with no employees apart from the founding team to more mature scale-ups with up to 30 employees. The companies were selected in order for each company to add a new dimension to the study. All companies operate in differing markets, with their own unique products, hence there are various environments and conditions which add to the value of the study.

Company	Main product	Number of employees in tech
A	B2C Digital media service	3
B	B2C Health and Fitness app	4
C	B2B Diagnostics service	17
D	B2B Data analysis platform	6
E	B2B Image processing tool	5
F	B2C Food delivery app	11
G	B2B Digital Marketing tool	15
H	B2B Digital Sales tool	6

Table 3.3.2: Interviewed companies

The interviews consisted of 5 quick-fire questions aimed at mapping the interviewee's perceived awareness of TD and to what degree they actively worked with TDM in their daily work. The quick-fire questions were followed by 8 questions aimed at collecting contextual information about the company, its product line, and its competition. Finally, 5 overarching topics were discussed, aimed at understanding the company's

development process, decision-making process, team structure, previous and present implications of td, and readiness for future growth. For the complete structure and motivation of the question, view Appendix A.

3.4 Data Analysis

This section introduces the theory on which the analysis was based. Furthermore, the section describes the course of action when performing the analysis.

3.4.1 Qualitative Analysis

The nature of qualitative data, in contrast to quantitative data, is that the analysis is performed through conceptualization, not diagrams and statistics (Saunders 2015). For the qualitative data, the concept of thematic analysis has been applied to have a systematic and flexible theory (Saunders 2015). Thematic analysis is flexible partly because it is not tied to a specific philosophical position, and partly whether it is used in an exploratory or explanatory analysis (Saunders 2015). Furthermore, it is flexible in regards to the approach taken, where this study has had an abductive approach, which in turn derives themes from the data gathered (Saunders 2015). The procedure of performing a thematic analysis is often concurrent and recursive during the steps (Saunders 2015).

The first step of a successful thematic analysis is familiarizing yourself with the data acquired (Saunders 2015). This is done through re-reading, transcribing, summarizing, and checking the data multiple times. This happens continuously during the study, as the data is reviewed several times during the study, for knowledge-gathering purposes, as well as controlling that the summaries and results are correct. Whilst familiarizing yourself with the data, as a first step of tangible actions of processing the data, coding of the acquired interviews is initiated (Saunders 2015). Coding is the process of labeling each unit of data within a data item, with a type of code or label that reflects the data's meaning (Saunders 2015). Since qualitative data is often complex and large, the purpose of coding the data is to make the items easily accessible, simplify comparison between data items and comprehend the meaning of the data (Saunders 2015). The codes used in this thesis were data-driven, which implies that the labels were created by the authors during the review of the data, based on the

purpose of the study (Saunders 2015).

The following distinct stage of thematic analysis is recognizing which themes permeate the interviews, and adjusting the initially proposed themes accordingly. This is done continuously during the analysis and processing of the data. The last stage is to finalize the themes and complete the analysis (Braun and Clarke 2006).

The process was performed continuously throughout the study. In a push to begin the implementation of the study, 4 interviews were conducted and transcribed from the video recordings. The interviews were then proofread from both authors, to ensure that the transcriptions were correct, as well as familiarizing themselves with the data. From these interviews, the initial themes were established and processed.

Throughout the first interviews, seven main themes were discovered to explain the origin of TD in start-ups, how this is managed and used strategically, and how this translates into the further scalability of the product. These themes were set to form a basis for the analysis of the empirical results. During the study, and the processing of said themes, it was concluded that a more condensed version, including five themes that could explain the generalizable parts of the results, was better suited for the analysis. The five themes that developed during the interviews were: *Vision*, *Process*, *Competence*, *Prototyping*, *Pivoting*, with the coding tree presented in Figure 5.0.1.

These five themes are represented by a section in Chapter 5 - Results & Analysis. In addition, the sub-themes identified are represented in subsections of the respective themes, to further add to the subsequent discussion, and to create a solid structure from which proper analysis may be conducted.

The themes and interviews were iteratively processed during the study and in conjunction with further conducting of interviews. Thanks to this, the themes could be further analyzed, reviewed, and adjusted as more data was collected. Furthermore, the analysis and review of the data were performed individually by the authors, which aided the analysis since multiple personal conclusions and themes had been drawn, which created rewarding subsequent discussions.

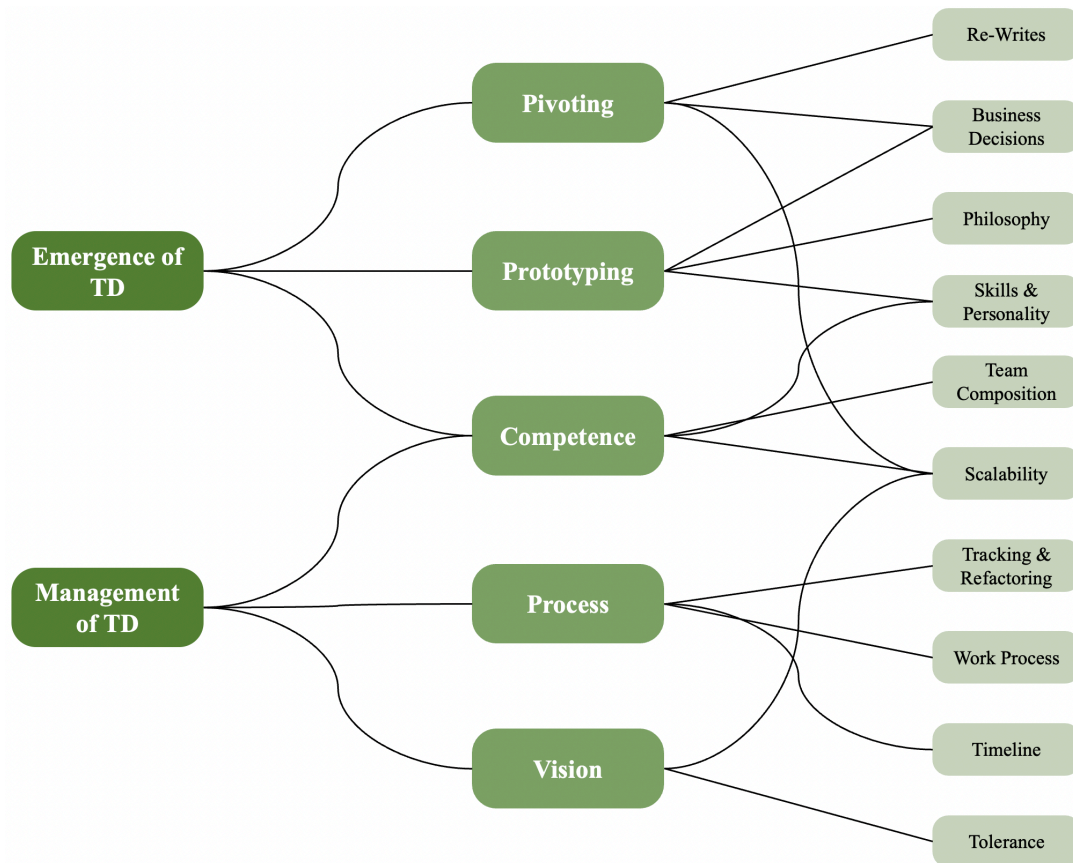


Figure 3.4.1: Code Tree - All themes Included

3.5 Rigour & Ethics

In any thesis, the rigor and quality of the study are of the highest importance. Without taking action to ensure that the quality and rigor of the study are in place, the research may turn out null and void. To ensure that the trustworthiness of the work is indisputable, qualitative researchers make use of formal and systematic methods for data collection and analysis (Shah and Corley 2006).

Furthermore, there are established measurements of rigor and trustworthiness in qualitative research (Shah and Corley 2006). The study must adhere to and intend to reach a strong stance in these measurements to make sure that the study could be considered trustworthy and reliable. *Credibility* is achieved in this study through the triangulation of data, by ensuring that the results gathered from the interviews are valid through the quantitative analysis. Moreover, continuous peer debriefing during the study has been performed. The data has also been analyzed independently to ensure *Transferability* is taken into regard through thorough descriptions of methods, interview questions, and transparency with included data. *Dependability* is handled

by letting the interviewees stay anonymous, and through purposive and theoretical sampling. Lastly, *Confirmability* is ensured through meticulous recording of data and results, and transparency with notes on theory and methods. This increases opportunities to duplicate the study to increase its validity.

Regarding the ethical considerations of the study, it is important to adhere to good ethics to ensure no harm. To protect participants from exploitation and potential harm to get the informed consent of each nuance of their participation (Bell and Bryman 2007). This has been ensured through active specific inquiries that the participants have responded to. Proper and meticulous citation, to give credit to the authors that have created the theory or analysis, is important to take into consideration to ensure that the study does not gain credit for others' work.

Seeing as how the supervisors of the study, CodeScene, provide a tool that aids in the tracking of td and could benefit from a study that concludes the perks of using their products. To counteract the possible ethical complications this brings, both the authors and the company has taken action to not influence the outcome of the study in a compromising way. CodeScene has not made any demands on what the research subject ought to be, and has not attempted to guide the authors towards any form of biased directions or conclusions. In turn, the authors have made sure to act objectively during the data gathering, as well as not letting their analysis be influenced by personal opinions and goals. The empirics have been approached individually, and objectively, to draw valid conclusions based on the evidence gathered.

Chapter 4

Context

This chapter aims to provide some context to the value created by, the role of, and requirements to succeed of start-up companies in general and start-up companies in the Stockholm region in particular. Hopefully bridging the gap between theory and reality, further motivates why means of facilitating early-phase start-ups is not only beneficial to individual companies but also to a wider entrepreneurial ecosystem, the prosperity of the local region, and the economy.

4.1 The start-up ecosystem of Stockholm

The existence of high-growth companies in a region can be strongly related to the quality of its entrepreneurial ecosystem (Stam and Ven 2021). Therefore, It could be interesting to give some context to the start-up ecosystem of Stockholm, the region this study has focused on.

Entrepreneurial ecosystems usually contain networks of entrepreneurs, finance, talent, knowledge, and support services (Cavallo et al. 2019a). These are all elements important to sustain entrepreneurship in a region (Stam and Ven 2021). Stockholm is often referred to as a hub for start-ups, both from a Swedish and also European perspective. The city is the birthplace of multiple successful start-ups, such as Spotify, Klarna, and Skype (Hammarberg 2020). Sweden can be considered one of the most competitive countries in the world, known for its favorable business climate and innovation drive (Balawi and Ayoub 2022). Further, the entrepreneurial ecosystem of Sweden ranks high on the Global Entrepreneurship Index (GEI), sixth globally and

third in Europe (Balawi and Ayoub 2022). Furthermore, in the annual Global Start-up Ecosystem Index report created by StartupBlink, Stockholm ranks fourth among the European countries, and 23:d on the global ranking, with Sweden ranking 5:th globally among countries, ranking second among their European counterparts, only falling to the UK (StartupBlink 2022). Sweden has advanced in the rankings over the latest years, further attracting capital and competence to the region.

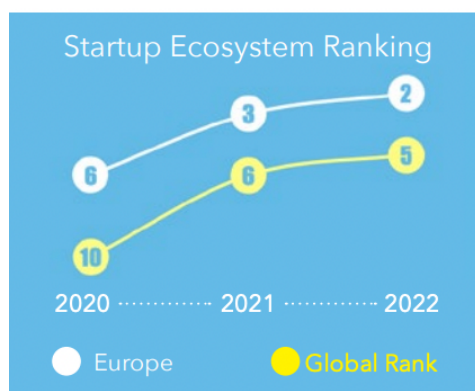


Figure 4.1.1: Swedens Start-up Ecosystem Ranking (StartupBlink 2022)

Furthermore, Sweden ranks first in the "business score" part of the report, leading giants such as the United States and the United Kingdom (StartupBlink 2022). This score is based on conditions influencing the business environment of an area, such as national infrastructure, policies, and legislation, which includes internet speed and freedom, top universities in the location, a diversity index, and patents per capita.

Around 5% of Swedish adults have started companies and around 9% of the working population is self-employed. The latter figure is lower than the average EU nation. However, these entrepreneurs have been reported to produce more new products and services than the average entrepreneur across the whole EU (Balawi and Ayoub 2022).

Of all venture capital invested in Swedish companies between 2017 and 2021, 75% was invested in Stockholm-based companies. In addition, Stockholm ranks 5th among the European cities that attracted the most venture capital in the same period (Stockholms Handelskammare 2023)

4.2 Economic impact

Start-up companies' effects on the economy and regional development have been well studied and found to be important for development and growth. Entrepreneurship can be considered the engine of countries' economic growth (Cavallo et al. 2019a). Further research has shown that young companies are a driver of job creation and that high-growth start-ups accelerate the reallocation of jobs from old to new industries (Stam and Ven 2021). A case study of Swedish start-ups and their effects on local development found that start-ups positively affect employment growth at the municipality level, and also positively affect population growth (Westlund et al. 2011).

Tech start-ups rely on funding from VC companies and angel investors to enable the growth of the company (Cavallo et al. 2019b), according to a report on the Swedish tech ecosystem conducted by the Swedish Institute, 2021, Sweden's ability to grow the interest of VC capitals and increase the funding received, further improving the conditions to establish a successful start-up.

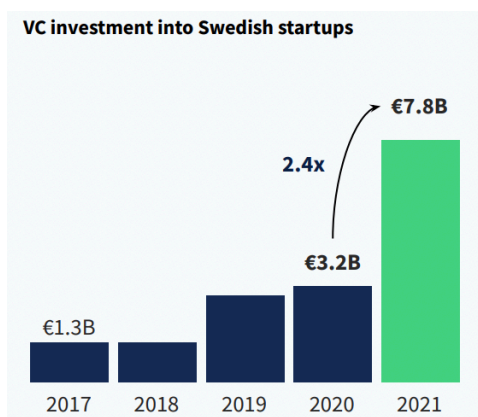


Figure 4.2.1: VC Funding Growth 2017-2021, (Institute 2022)

4.3 Societal impact of start-ups

Sweden's start-up scene capitalizes on the unique advantages within the country, such as a strong business environment, gender equality, and a high quality of life (StartupBlink 2022). This aids in attracting international talent to start their businesses in Sweden to take advantage of the benefits and opportunities presented, which in turn aids in bringing diversity and creating new job opportunities.

Tech start-ups operate in many fields, plenty of them impacting societal functions. Examples such as Education Technology (EdTech), which creates new opportunities for learning and education. HealthTech, such as Kry or Doktor.se has revolutionized healthcare practices in Sweden, enabling health counseling and distance care. Tech start-ups such as these are major contributors to the development of society and accelerate the innovation landscape.

To summarize, the conditions to build a successful start-up in Stockholm are far above average. The business connections and possibilities, infrastructure, and a large amount of interest and talent contribute to making Stockholm one of the most attractive cities in the world for start-ups. The effect on society that start-ups like Spotify, Kry, and Voi have had are massive. Since the conditions to create a successful start-up are present in Stockholm, the aspect that may jeopardize the company lies in the execution. For some products, there isn't a market yet. But for some, their contribution to the development of the market may be hindered by technical difficulties. To handle these crippling difficulties, such as TD, further studies on TD and similar areas are needed to set the best technical conditions possible.

Chapter 5

Results & Analysis

This chapter presents the empirical findings from the semi-structured interviews. The results are presented based on the themes found throughout the analysis of the interviews, where each theme will represent a section in the chapter.

5.1 Vision

"The vision has been pretty clear from the start and there are a lot of those principles left. I don't want any technical debt. (I-07)

A theme that was noted and developed during the interviews was the respondents' view of the importance of the company's overall vision when developing the product. Many of the interviewees referred to their company vision when explaining how each company views TD, and how they operate surrounding the subject. Furthermore, they related the short-, long- and holistic vision to the occurrence and avoidance of TD. When the vision of a company was clear and concrete, and included a long-term idea of where one was aiming to be, managing TD often becomes included in the collective development, and preventing its occurrence comes practically as a side effect of the set goals (I-02, I-03, I-04, I-05, I-07, I-09).

"Basically, we were aware that we started out small, you are going to need to progress, step back and iterate, but that you are looking at the grand vision in combination with. Here I think it's about personality traits and personality. Our CTO is very thorough and wants to think as many steps ahead as possible." (I-02)

Interview respondent I-02 declares when asked if they were actively working with TD from an early stage, that it wasn't the fact that they were actively taking TD into their consideration, but that it came as a side effect of their overall goal along with their CTO's inherent personality. This is further emphasized by him when asked about how they plan to manage the TD accrued before a scale-up phase.

It is not exactly spoken that we need to have a good grasp of our technical debt, but it is de facto the case that we have. Otherwise, our business case and vision would lose too much. I would say that if you were to sort it out, then it is a very important part of our core vision. (I-02)

Furthermore, the vision helps create a common target for every employee to work towards. It is found that having a strong and clear vision to strive towards has a positive effect on TD, as it simplifies for the tech team to make well-reasoned and calibrated decisions. The tech team is also more prone to accept feedback and implement the correct changes, instead of jumping on every point of feedback gathered, with a concrete and tangible hypothesis of what they are supposed to do. (I-02, I-03, I-04, I-05, I-07, I-08, I-09)

Another thing is to concretize the vision which I have been working a lot with the last quarter. Not only to sell the correct product to our customers but to give the tech team something to adhere to in their decisions ... So I think that the vision that now has become more concrete aids in us having proactive discussions. Incredibly much. (I-04)

However, establishing and sharing a solid vision for the company to adhere to needs to be done with finesse. If the vision is thrown on the teams without direction on where to go next, the vision could complicate the processes. Interviewee I-05 explained how the vision had changed and during its solidification, the employees were gradually included in the vision of the company, where he explained that it helped him understand the value he created and ensured that he could guide the team to make decisions in line with the company's vision. When asked if he felt that it helped to be included in the discussion around the company's vision in their everyday planning and work, he answered.

I feel like it does. But then the thing is, it's kind of like a double-edged sword, right? So it can help, but it can also be a problem. It does help to some extent because you know what sort of things you're going towards so

you can avoid issues in that direction. But then the thing is, sometimes you might like to overthink things and might do the opposite. So you might be slowing yourself down by overthinking. So you have to find a sweet spot with it. (I-05)

To be able to more effectively ask the right questions considering business decisions, and to work towards a solid vision directly impacts the scalability of the product. If the long-term vision is one of scale, and the aim is clear and well communicated to the tech team, the decisions and discussions may always be made with the long-term vision in mind, effectively reducing any accrued TD before a scale-up phase. (I-02, I-03)

5.1.1 Visions Impact on Scalability

When considering the effects that the vision has had on scalability, several of the interviewees mention that when the vision has been lacking, larger parts of the code have been unusable and rebuilt quickly from the finished product. When the vision hasn't been in place, solutions, and business decisions have been made with a problem-to-problem approach, resulting in recently refactored pieces being rebuilt. (I-04, I-05, I-07)

Given where we are now with the product and when we also want to scale it up and broaden the offer to more clients, a bigger customer base, we see that we have a lot of technical debt that we need to deal with. From the vision that has been recently updated from where we want to be with the product in a couple of years. Then we need to rebuild some parts that have just been rebuilt. (I-04)

In contrast, when the vision has been solid from an early stage, and the possible expansions and extensions have been considered and included in the vision, the interviews show that less TD has occurred as a result of needing to alter the product to adhere to scalability. (I-02, I-03, I-04, I-06, I-09) Furthermore, when asked about their choice of systems and programming languages, it showed that when the leaders and developers knew what the outline of the scope was the choice was made easier since they could make an informed decision in line with the scalability of the product (I-03, I-05, I-06, I-08).

In the case company of I-08, their vision was different, since they worked via the motto

”do things that don’t scale”. The reasoning for this was to quickly build features and products, release them in the market, observe how the market responds, and then act accordingly (I-08). This is a common practice for most of the interviewed companies (I-01, I-02, I-03, I-05, I-07, I-08), however, it seems to be seldom it is taken to this extreme. In this way, TD is quickly built up and is further complicated since this may have the added effect of TD items having implicit dependencies, which creates an exponential increase in the time needed to manage and work around the TD when developing the product (I-08).

”Yes it is absolutely chaos on chaos. Technical debt makes it hard to add anything and you have to get it done. You chop it up, even more, to add that thing as well. It really becomes exponential. (I-08)

Therefore, if you have a vision that does not adhere to the scalability of the product, TD items are bound to increase. However, even for the extreme case of the ”don’t build to scale” mantra, a long-term vision has been a necessary development to ensure the scalability of the product (I-08). Financial aspects of the business created the need to stop ”burning” money and establish a foundation on which to scale. This led to the I-08 company establishing a midterm vision, and when asked if he believed that this aided in their management of TD, he answered;

”100%. Before Q4 last year, the thought was to build something scalable. Then the technical physicists and mathematicians began over-designing things. I then pushed for a concrete vision during Q1 and Q2 with mock-ups and everything. So now it is easier to know when to invest in technical debt, and when not to. (I-08)

Nearly each interview subject has hence described implicitly that a good vision has aided them in managing and addressing TD. This is by aiding the developers and C-level executives to make educated decisions with the scalability and goal in mind, reducing the occurrence of TD by not building components that are not included, or may harm the vision (I-02, I-03, I-04, I-05, I-07, I-08).

5.1.2 Technical debt tolerance

Different attitudes towards the build-up of TD among the interviewed companies were identified. In most cases, the vision had a tangible effect on how much TD was

permitted to be accrued. This had an impact on business decisions, pushing deadlines, fast releases to the market, and overall development. For some, the tolerance was decided by their ability to afford to accrue TD, but also their ability to afford to handle previously accrued TD, as there are often strict funding agendas. When it comes to continuous management of TD, many of the start-ups feel like they can't afford to manage much of the accrued TD, hence the tolerance for already accrued TD in non-critical functions is often pretty high (I-01, I-02, I-04, I-05, I-07, I-08). However, when the vision includes restrictions due to low funding, the tolerance of accruing TD may be significantly lowered.

"When you put it in relation to that it is mostly our own funding, and friends and family, then it translates in a way that we can't afford to take shortcuts and create technical debt that we can't handle. We also can't afford, which is something quite usual for companies but unspoken, to take shortcuts at a certain stage because you think that "if we just get to this point we unlock funding to tackle our current problems." We have not taken that approach." (I-02)

The technical debt tolerance is hence influenced by the vision, and how the vision impacts the core values of the development teams and product owners. For the previously mentioned "don't build to scale" mantra, the tolerance is significantly higher. Here, the system is built around putting out fires, but delivering quickly (I-08). However, the costs of TD are increasing at such a rate that tolerance is being reached as well. For other companies that are self-funded or have a low amount of external funding, the trade-off of cost to immediate value for the time spent by the employees often lowers the ability to handle TD, which implies that the tolerance needs to be set higher (I-01, I-04, I-05, I-07). Whereas when the financial ability to reserve funds to be used on managing TD presents itself, the aim may transfer to preventing TD, and the tolerance may be lowered as an effect.

"We now afford to spend the time on not building more technical debt, but instead even prevent technical debt much more moving forward." (I-07).

The competitive landscape of the companies interviewed was also identified as a contributor to the TD tolerance and how they developed the vision. The competitiveness of their respective market was found to impact their reasoning regarding product decisions and priority orders. When the market was highly

competitive, a primary focus was to develop the product and deliver it to the market swiftly, thus accruing TD by cutting down time-to-market (I-01, I-03, I-05, I-06).

“Yes what we are building right now is very risky when you have players like Spotify that can do it in an instant. It has always been that we want to release it as quickly as possible.” (I-01)

To summarize, the results from the interviews conclude that the effect that a solid vision has on the creation of TD is tangible. For companies that have used a vision that included how they would scale, their TD tolerance, and the core value of the product shared with the entire teams The effect has been that C-level executives may take more holistic business decisions, and the development teams are able to produce code that take in to account where they want to be in a few years time, and what the end product is. By establishing a concrete vision in an early stage, a plentiful amount of TD is able to be avoided by building desired modularity, taking architectural decisions that are customized for the applicability of the vision, and not building technology that will not be used. There is often a risk with not having a vision, that a large percentage of the product built is not in line with later decisions, leading to wasted resources on products that may never have been tested, in an environment where there are often limited funds.

5.2 Process

The development process and timeline, as well as the overall work process, have varied between the respondents. start-ups are overall dynamic and need to be agile and swift when adapting to new trends, and the environment of the start-up is often ad hoc and fluid (I-01, I-02). Hence, many different versions of the processes were found in the interviews, with differing approaches, best solutions, and best fits.

5.2.1 Development Timeline

The development timelines of the different companies interviewed differed and were mainly found to be influenced by factors such as funding, business opportunities, and market conditions. Multiple companies expressed that they had been able to develop their products at a relatively smooth and steady pace over time (I-01, I-02, I-03, I-05, I-07). However, in some cases, business opportunities and customer demands affected

the development timelines and demanded rapid development cycles in order to meet deadlines (I-02, I-03, I-06). Another factor that was found to affect the development timelines of companies was funding and the ability to work full-time on projects. Multiple companies interviewed started out by running their business as a side project, spending time and performing sprints when time and resources were available (I-01, I-02, I-04, I-07, I-09). This limitation was found to affect the overall development- and decision-making process.

The management of technical debt was also found to differ between the stages of the start-ups. As described earlier, TD tolerance is higher at the inception stage and more TD is accumulated. When the proof of concept is established however, TD becomes a more hindering problem, and there is often a need to invest further resources into nullifying the potential obstacles that occur because of TD (I-01, I-03, I-04, I-05, I-07, I-08).

”Because by this time, we started to feel confident about our way of doing things. And then we realized that, OK, now is a good time to spend extra dev effort into making this more future-proof.” (I-03)

Furthermore, when start-ups reach the phase where they start to consider raising funds through external sources such as Venture Capital firms, the conditions for the future development of the product change. These events often call for complete re-writes, where the resources are put to better use by completely focusing on the new version, instead of trying to adapt the old version to the new conditions (I-10)

When transitioning into a more stable phase, where the majority of the companies start to truly consider scalability, TD is often included in the discussions. When a company starts planning to scale up and further develop its product, TD often creates more obstacles than in previous stages. This indicates that the impact of TD becomes more tangible the further a company progresses. Hence, the previously acquired TD, which is often not impacting the development at an early stage, starts to create tangible, negative effects on development. A firmer grasp of where the TD has been acquired at an early stage, and discussions on how companies should work around it would therefore be beneficial. Where solutions to early accumulated TD may consist of complete re-writes which will be further discussed later, or being included in the vision and company culture.

5.2.2 Development Process

Two main types of development processes were found in the interviews. The first was more or less unstructured project cycles where team members worked largely independent on one another, with less focus on process and project management tools (I-01, I-02, I-04). This development process was by and large adopted by companies with small development teams or with development teams largely consisting of part-time employees or consultants. This development process was highlighted as the most effective by the companies using it, given their current resources and daily operations.

"We are not that process-oriented, it becomes a trade-off between spending time documenting, writing guidelines, reviewing work, etc. We are a small team and work very independently of each other. I think in a sense that is how a start-up needs to operate, you cannot work like a large corporation." (I-02)

"We are working agile, iteratively. However, we do not follow a fixed process. Mainly because it would not benefit us given our current situation. We would have benefited more from it if we had more development resources ... We use a kanban board and try to keep the amount of work in progress as low as possible but try to focus on solving the right problems at the right moment." (I-04)

The second type of development process consisted of more formal, structured, and scheduled project sprints. This kind of development process was mainly employed by companies with larger development teams (I-03, I-05, I-06). These companies organized their work through scheduled sprints, ranging from 1-3 weeks in duration, daily stand-up meetings, and recurring retrospectives.

Moreover, it was found that companies had a tendency to move towards more structured development processes as they matured (I-04, I-07, I-09). Partly due to the increased number of employees, but also due to the increased complexity of daily operations.

"As we have increased the number of products in a production environment, we have learned the importance of structure and process."
(I-07)

Another company found that as their product and technology matured, project scopes could be kept much smaller since fewer components had to be built in parallel for each project (I-09). As project scopes could be kept smaller, and tasks more clearly defined, more clearly defined procedures and project sprints started serving the company better than ad hoc planning and project management.

5.2.3 Technical Debt tracking

A common theme in the interviews was the way companies kept track of their TD. None of the companies interviewed used dedicated TD tracking tools such as static code analyzers. Instead, most companies interviewed claimed to track and document their TD through semi-structured processes such as creating tasks on a project backlog board (I-01, I-02, I-03, I-05, I-09). In addition to tracking TD through backlog tasks, much of the tracking was found to rely on the system knowledge and experience of the developers at the companies (I-01, I-02, I-03, I-04, I-07, I-09). In other words, to some extent, relying on developers "knowing" what areas of the code base are problematic and what implications that has on further development.

"We handle it somewhat ad hoc, and keep in the back of our minds that we should improve something, but keep it for a later time ... So it is not something we have a structured process around, it is something we do when we feel like it is the right time to do so." (I-01)

These personal dependencies were found in some cases to cause issues for the TDM activities at the companies. Both in companies with small development teams consisting of part-time or short-term consultants, but also in larger, more structured development organizations. The main concern for small development teams seems to be problems regarding personal dependencies and knowledge transfer between new hires and part-time consultants. One interviewee expressed concerns regarding the company's ability to onboard and convey system knowledge to new hires given a low degree of documentation of their system's TD (I-01). This problem was shared with another company, that had made new hires that were unaware of where TD was located in their code base:

"If we take into consideration the bits of code which nobody has looked into for months or even years, which might pop up once we have new clients coming in, those I might be afraid of because I don't even know what I'm

getting myself into.” (I-05)

A third example of this was an interviewed company that was forced to do major refactoring work after external consultants had worked on a project without proper design guidelines or ways of documenting TD (I-02). Further, multiple interviewees expressed that they were aware of the shortcomings of their ad hoc TD tracking strategies (I-01, I-02, I-07). However, they emphasized the fact that more rigorous and structured processes would consume additional resources, and these processes would only yield value if enough resources are available to follow things through (I-01, I-03, I-07).

”The reason we work the way we do, is because we do not have enough dedicated development resources. Perhaps we should use better tools for tracking TD. However, no one on the team has the time to run and operate that project, meaning that it would only create unnecessary overhead and administrative work.” (I-07)

In addition, the notion that code that is built early on might not survive more than a few months, due to pivots and future re-writes, was also found to influence companies’ commitment to thoroughly track and document their TD (I-01, I-02, I-05, I-07, I-08). Mainly in the way that it leads companies to accept more TD in parts of their code base.

”I think we are aware of some risks, but we don’t focus too much on mitigating them. There is a possibility those risks won’t be relevant anymore, in which case it doesn’t really matter.” (I-01)

Slightly different problems were also found at a company with a larger development organization. At this company, routines for documentation and tracking of TD were in place. However, the growth from one to multiple teams caused problems in sharing knowledge about where TD was introduced as developers from different teams contributed to the same repository (I-03).

”Because when you split teams, you have the same code base that you’re contributing to. And in our case, we don’t have clear ownership of the code base ... Then it’s easy for you to create TD for other teams.” (I-05)

5.2.4 Refactoring strategies

The main type of refactoring strategy found in this study was ad hoc and necessity-driven and was employed by most companies interviewed (I-01, I-02, I-04, I-05, I-07, I-08). In these companies, refactoring of problematic system components was often conducted when the friction of TD reached a perceived pain threshold.

None of the companies interviewed had recurring project resources allocated for refactoring and maintenance of their software or architecture. The most structured refactoring strategy found was companies that had kept TD items on their planning board and therefore also had the ability to prioritize and incorporate TD repayment in their ordinary planning process and project sprints (I-03, I-05, I-09).

An insight encountered in one of the interviews connected the awareness of TD to the legitimacy of refactoring strategies (I-08). The rationale was that spending time refactoring code can be difficult to motivate unless there is an awareness of TD in a code base and the friction it creates in daily operations.

"First of all you have to be aware of the existence of technical debt, and that it actually consumes x hours each week. In that case, it's justified to incrementally refactor the worst parts. But if some people don't see the technical debt, or are not affected by it, it might not be received well." (I-08)

To summarize, companies were found to operate differently in regard to the development process. The resources available to the companies, as well as their maturity, were found to be key factors of this discrepancy. Small companies expressed that they could work more effectively with ad hoc development practices and less overhead from strictly enforced procedures. Limitations in regard to full-time commitment and part-time employees were also found to influence such considerations. Further, it was found that companies move towards more structure and better-defined processes as they grow and mature. This is partly because of the larger perceived impact of TD in a scaleup phase. To what degree companies actively work with tracking their TD varied. As stated previously, none of the interviewed companies used specialized tools to do so. Lack of TD documenting processes was found to increase the dependency on developers' knowledge, even though this might be an issue even for companies with existing tracking practices.

The results presented in this chapter are not sufficient to draw conclusions as to what development process incurs the most TD. However, ad hoc development practices, mostly employed by small companies, leave wide gaps for TD to grow, without any control or tracking, other than the good memory of its developers. However, this was to some extent found to be deliberate risk-taking by companies, as some were uncertain about what parts of their system would survive long enough for TD to become a real issue.

5.3 Competence

A common theme identified in the interviews was the importance and impact of the team members and their competencies. Both on the emergence of TD in software projects, but also the management and reduction of TD. It was found that not only is the personal knowledge and competence in writing good code of large importance but also the different personalities and team compositions (I-02, I-03, I-04, I-07, I-08). The importance of how the awareness of TD with at least one lead person within the team is found to be a key component among several of the companies, which will be visited further in section 5.3.2.

”And the problem is that everybody approaches it as a planning problem, which, you know, if you do an MBA, it makes sense. But in real life, you need people who actually are good at that and who are good at thinking about the platform, thinking about the fundamentals and, finding all of these areas of imperfection and then handling that.” (I-03)

The in-house competence and personal skill set were also a strong contributor to the occurrence of TD and architectural decisions which would impact the scalability and TD. Nearly all interviewees stated that in-house competence influenced programming language, architectural decisions, and management of TD, hence it influenced many of the critical early business decisions, making it a crucial part of a start-up’s development.

5.3.1 Skills

The skills and previous experiences of start-up founders and early employees were found to have a large effect on decisions made at these companies, especially at the

early stages, which is referred to in the literature as the inception. How decisions were made and for what reasons, was found to influence the amount of TD accumulated. For instance, at the inception phase of companies, as decisions regarding the first prototypes were made, most of the companies acknowledged that the first technology choices were mainly driven by skill set and previous experience, even though the technologies chosen were not necessarily optimal for their future scalability or optimal for their product overall (I-01, I-02, I-04, I-05, I-07, I-09).

"We had a data scientist that was used to work in R, which led us to build in R." (I-04)

"Well, that was just the initial people that we had, the very, very first prototype that we had. Back then, there was only Python expertise in the company." (I-03)

These kinds of decisions were also commonly found to be influenced by a desire to get going quickly and to some extent the comfort of the developers (I-01, I-02, I-03, I-04, I-07, I-08). Developers preferred working with technologies they were already familiar with, in order to ensure the quality of what they were producing as well as the ability to produce features faster in early development. Given these incentives, combined with the constraints of the founding teams' skill sets, early technological paths were in many cases taken with shortsightedness which in many cases proved to introduce TD early on.

"There was a combination of reasons. At first, we decided to go with something we already knew. As time went on we had become better at this other framework, and felt it was the better alternative, partly because it was a much more popular language and had a lot better ecosystem." (I-01)

When the time then comes to scale up, the TD accrued must be refactored (I-01, I-07, I-09). This often results in complete re-writes of core features, where the framework or system will need to be transferred to another programming language, or other major system transfers (I-01, I-02, I-04, I-05, I-07, I-09). If the required competence is not currently in the company at that point, the solutions are often to employ new personnel, hire a consultant or learn the new systems (I-01, I-02, I-03, I-04, I-05, I-06, I-07, I-09). The latter is often more time-consuming but generates new competence where the core values and vision are already imprinted (I-01, I-02). It also takes time to gain

expertise in the area, hence, poorly written code may stem from this in the early stages of rewrites.

"The first decision was to use what we knew our competence was. Then when we changed, we received better competence in the other language that we had developed at a later stage. We also saw that it was a more popular language. So we thought that it might be better to connect to a language where the ecosystem is better." (I-01)

Employing new personnel fills the competence gap, but is often costly. It is also quite a commitment for start-ups with limited resources. Each start-up would of course want to have 100% competence coverage, but it is a prioritization question in the form of what competence the company may acquire with their funds (I-02, I-04). Hence, to gain the needed coverage in-house, larger investments might be needed.

"Of course, you want a development team that has all the competencies we need from the start, but we have had to be smart and acquire what we need most at the moment. We have probably known all along that we have needed the competencies that we have now, but have been forced to choose the most important competencies for where we are." (I-04)

Technical debt is then accrued through not having all the necessary competence needed anyway, having to create sub-optimal solutions, and making decisions that the company is not mature enough for (I-04).

"What has hurt us is that we have had to sometimes cut corners in how we build things. Furthermore, we have made decisions that we have not been suited to, competence-wise." (I-04)

Hiring consultants yields quick and effective results (I-02, I-06). However, these consultants often develop software with a dedicated problem at hand and don't necessarily comprehend the full picture of the scope of the company. Furthermore, since they are on an hourly rate, they often need to provide immediate value, yielding quick and strong results, but often with TD built into them (I-02, I-06). The Interviewee I-06 has had previous experience as a consultant, and explained the effects as the following:

"I sell eight hours a day to you. I can be productive and create new features all these eight hours, but the more technical debt I build, the more

I buy time now. In the future, these eight hours won't be as productive, as I will spend an hour repairing things that are technical debt. Then I will only work seven out of these eight hours while you still pay for eight, and thus my hourly rate will inflate in a way. When you convert it to money, then the clients listen.” (I-06)

Therefore, it is important to have strong guidance for the consultant, on not only what the task is, but also all the surrounding aspects that need to be considered when developing the product post the consultant's contribution.

”We had a consultant about two and a half years ago. Really talented, and extremely quick but not very thorough. It was a trade-off, we evaluated afterward and thought that we were very happy with what he had done, but that it created quite a bit of technical debt. Parts of the work he did we have had to revisit, refactor and optimize for performance. Right now we are even refactoring a part of it completely.” (I-02)

Hence, there are many aspects of competence within the company that influence TD in different stages of development. Adhering to comfort and speed is important in an early stage, and an early accumulation of TD is not always hindering but is often the better choice in the common value-to-cost trade-off. (I-01, I-02, I-03, I-04, I-07, I-08) However, a company should not linger too long before deciding what system is most beneficial for them, acquiring the competence needed, and refactoring enough to enable scalability. (I-07) There are evidently many forms of acquiring the required competence, with their own perks and negatives. The suitability for each option is often dependent on the situation of the company, and is not a "one solution fits all". Therefore, careful consideration and proper planning are required when choosing how to tackle the problem, and how to implement it in the best way.

5.3.2 Personality traits

Several companies mentioned the importance of the different personalities within the company with regard to TD. When a person has a knack for considering TD in everyday business and product decisions and communicated this to his colleagues, this had a greater effect on the prevention of TD than most planning (I-02, I-03, I-04, I-07, I-09). Thus, many companies considered the personalities of their colleagues to be highly important, an aspect that is not often considered when handling TD.

"I think a lot has to do with personality, I think our CTO is extremely thorough and likes to think multiple steps ahead. " (I-02)

Thoroughness and attention to detail were regarded as important personality traits of developers that partly explained why certain companies were able to keep the amount of TD low even though they lacked rigorous TDM activities in their development process. However, it was not necessary for the entire tech team to have these traits, but in the overall team composition, the trait was deemed critical (I02, I03).

"He has this thing in his head that he has to constantly improve everything behind the scenes and give a clean way for developers to use them. And so even without asking, he will just keep constantly improving the platform. And then there are other team members that are just, they are users of his work." (I-03)

Another trait that was identified to be important was humility (I-02, I-04, I-06). Humility in regards to not being overconfident in that what you have created is always the right choice, and that you are humble to be able to pivot and "kill your darlings" so to speak. This is important in the aspect of managing your TD when you know that there is a large TD issue that needs to be fixed in a pivotal way, but you are not willing to trash the work that you have accomplished so far (I-04, I-06, I-08). This will in turn add TD exponentially.

It so easily happens that you just keep on building on the technical debt that it becomes, maybe not gamblers fallacy, but that you just keep on throwing money on a semi-sinking ship because it is still afloat. For each part that you add, it only becomes harder to solve later on." (I-06)

Humility is important continuously throughout the development, along with thoroughness and attention to detail, as the product keeps evolving. (I-02, I-04, I-09)

"I believe that it is reasonable to refactor continuously and successively. The worst is if there is prestige in the decisions. Not make decisions about a certain solution and keep building on it when we know we should rip it up. It's my and the architect's job to convince management that it will cost a bit more now, but it will be cheaper in the end because we're not adding to the debt, we manage it early. ... The decisions were made from

the foundation we had then, and then it was right. But we have another foundation now and we know that it is not right anymore. (I-04)

However, it is not only the personalities in a team that matters, but also the way that the team is built and the diversity of the team to best handle the issue of technical debt.

5.3.3 Team Composition and Dependencies

The team composition and how it has developed was also identified as a theme during the interviews, as both an influence on the accumulation and prevention of TD and an influence on managing the TD (I-02, I-03, I-04, I-07, I-08). Certain personalities may "over"-design products, slowing the process down and adding unnecessary costs, then personalities that balance that aspect are needed (I-08). However, relying too much on just one person to take TD into consideration may result in TD via personal dependencies.

"But for us, it's also about the team composition. And in my past experience as well, that actually matters much more than planning, which is why most companies fail at handling tech debt in the right way. I think it's the team composition that they fail at rather than the planning." (I-03)

Furthermore, technical debt is often accrued when the teams have a high in and outflux of employees (I-04, I-05, I-07, I-08). When handing over the position to a new employee, from one that is put on another task or quits, the transfer of knowledge is seldom without flaw, leading to solutions in place being dismissed or rewritten for example (I-08). Furthermore, as presented in 5.2.3, start-ups commonly lack extensive documentation of their TD, but rather key people have the knowledge in their heads (I-01, I-02, I-04, I-05, I-07, I-08). This opens up personal dependencies and missing existing TD.

"Our development process is somewhat ad hoc, we do not codify things but rather keep them in our minds. If a new person joined the team they would probably have a hard time understanding certain things in our codebase ... " (I-01)

The smaller size of teams in start-ups, where many must take on a large area of responsibility, often infers that you have to work independently from one another

and trust each other's judgment (I-01, I-02, I-04, I-06). However, when it comes to trust is also often accompanied by a motivation as to why certain decisions have been made, and what the possible value and implication of these decisions might be, including TD (I-06, I-08). In addition, several of the interviewees identified that it was important for at least one of the senior employees of the tech team, such as the CTO, architect, product owner, or other tech leads to have a solid grasp of TD, and needs to influence the team in how they work around TD (I-02, I-03, I-04, I-06, I-07, I-09). In ways such as reprocessing the decisions on features before implementing them, reflecting on possible consequences, and designing them in a way that is in line with their vision.

To summarize, competence within the company plays an explicit role in the accumulation, prevention, and management of TD. Previous experience and competence in a specific language often dictate the choice of programming language and system choices. There is a consensus that at the inception phase, using the competence that you have in-house to produce a product that is of high enough quality quickly is preferred, over planning too extensively and choosing a system or language that may be preferable when scaling, but will take too long to implement. These decisions may cause missed market opportunities, which in turn may cause the larger early investments of time and money to be for naught. Therefore, it is preferred to accept TD at an early stage and deal with this at a later stage, by adding the competence needed. For some, they find that the importance of having a good team, with personalities that are passionate about TD, is more important in the management of TD than planning for TD. Good team compositions also have balancing features, preventing the company from focusing solely on managing TD, or adding new features. A well-handled team, in regard to TD, will include a senior member that is thorough, and bring a holistic view into the development. Including the vision in the decision, and influencing the team members to consider all aspects before implementing a solution, to suitably approach TD.

5.4 Prototyping

Both similarities and differences in the way start-up companies prototype their products were found in the interviews. Companies were found to approach the uncertainty of product prototypes and the emergence of TD in this product

development phase differently. Also, the ways companies approached product prototyping was found to influence the amount of TD and have implications on further development in a scale-up phase. It could therefore be of interest to the study to explore what drives decisions regarding product prototyping in start-ups.

5.4.1 Prototyping philosophy

Prototyping was found to be an important aspect of start-up companies' development process. The companies commonly valued the ability to quickly try product ideas and attempt to prove hypotheses (I-01, I-02, I-03, I-04, I-07, I-08, I-09, I-10). Multiple interviewees expressed a desire not to spend time and resources building technology for features that have yet to be proven.

"The first version we build was a bit quick and dirty, we wanted to build something that was good enough to take to the market and probe the interest for the product. If you go out with your product and notice that nobody wants this, there's no point in wasting a bunch of development hours on it either. So the first version was very quick and simple." (I-04)

This reasoning ties back to companies' processes and resource management. As outlined in section 5.2, early-stage start-ups were found to value agility and flexibility over structure and process as they lack the resources to follow through properly. The resources available to start-ups dictate the number of development projects that can be carried out. Most companies interviewed expressed that they would rather build and evaluate rough and half-finished prototypes rather than be thorough and work with a long-term perspective from the beginning, in the interest of maximizing their available resources (I-01, I-02, I-04, I-07, I-08, I-09, I-10).

"In previous projects, we have always benefited from building as little as possible. If we're talking about technical debt, it's not about taking shortcuts, but rather prioritizing what features, frameworks, and libraries are absolutely necessary. Is this something we should do now or later? We have decided to be very conscious of what we're actually implementing. In a way, we might introduce technical debt, when we're only completing features to 60%, and add the remaining 40% in sprint 2 or 3. However, at that time we've been able to determine if this is a feature we will keep or not, without having built a bunch of incomplete features

or quick fixes.” (I-09)

In early-stage start-up companies’ desire to prototype product features rapidly, shortcuts were commonly found to have been taken. These shortcuts were often conscious and deliberate decisions, as companies valued getting a product on the market and acquiring proof of concept (I-01, I-02, I-03, I-04, I-07, I-08, I-09, I-10). These actions are all contributors to the occurrence of TD and commonly set companies on sub-optimal technological paths. As previously stated, the accumulation of TD at an early stage is often conscious. The notion that you know that what you are building is short-term, but because of limited resources, as it often is in start-ups, that is a choice that has to be made (I-02, I-07, I-08).

”I think to a quite large extent that there has been an awareness and we have taken decisions that we know implies shortsightedness in what we are building. We know it. No one enjoys it, but because of profitability, we need to make that decision now. Pretty often it has not been a technical, but rather a management decision that we need to advance. There aren’t enough time or resources to assign a month to build this, or send someone to educate themselves to build this properly.” (I-07)

One of the interviewed companies went further than most in their pursuit of rapid prototyping, completely abandoning scalability in their vision:

”We have had the principle to ”Do things that don’t scale”. Simply because we want to rapidly develop features and see how the market responds to them ... It’s been a highly deliberate strategy. We don’t know if the features will work or not. It’s easier to build them and see if customers like them or not.” (I-08)

Even though other companies interviewed did not have such an explicit short-term prototyping strategy, the outcome share similarities with other companies that will be covered in the following section.

As to when companies started shifting from a prototyping to a scalability mindset differed. Ideas of when an ideal time to do so also differed. Some companies expressed that such a shift occurred once they had found the right ”recipe”, and started feeling confident that they had a product that customers were willing to pay for (I-03, I-04, I-05, I-06, I-07). Another scenario was companies that wanted to wait until they reached

exponential growth, or until they had been able to raise capital (I-08, I-10).

5.4.2 Implications of early decisions on future scalability

The desire to rapidly prototype new product features, coupled with the technological considerations of prototypes, often resulted in implications for future development and the scalability of products. A common case was that technologies that enabled rapid development in an early stage, at a later stage became a limiting factor (I-01, I-02, I-04, I-07).

"We have run into scalability issues. Partly due to our frontend that was built in a framework that enabled us to prototype very quickly with little effort, but that doesn't scale well." (I-04)

Further, shortcuts on an implementation level also caused issues for some companies, mainly by creating friction in maintaining existing code and developing new features (I-03, I-08).

"We have multiple systems in production that depend on each other in implicit ways. If we want to make changes to the system we have to make sure that we don't cause any downtime for customers. So it can handling technical debt can consume hours in day-to-day operations." (I-08)

The limitations of early decisions often resulted in major re-writes for companies down the road. Usually due to a shift in priority order, where iteration speed was given a lower priority in favor of scalability and performance (I-02, I-03, I-04, I-07, I-08). In most cases, companies felt that it would be more effective to start over with insights learned along the way, instead of trying to improve the original prototype. This shift in prioritization also commonly resulted in new technological considerations, leading to shifts to technologies other than what was originally prototyped. How long these rewrites took to complete varied from company to company. However, weeks or sometimes even months were common timelines for such projects (I-01, I-04, I-05, I-07, I-09, I-10). These were in many cases resource-intensive projects that hindered the further development of products as they were ongoing.

"Right after we switched framework, there was a period where we could've developed new features instead of fixing bugs in the re-implemented features." (I-09)

Even though companies had diverted considerable resources from other projects to start over with new technology, most companies expressed a conviction that it was the right decision and did not regret having started out with their prototypes.

“We felt that we might throw away two or three weeks on rebuilding from scratch. So the sunken cost wasn’t that bad. In addition, it takes so much longer to build, design, and iterate initially than it does to just rebuild the technical parts when you know what you’re building from the start.” (I-10)

One interviewee even expressed that the fact that they had started out with simplistic prototypes that later had to be re-written, resulted in lower amounts of TD than if they would have tried to build everything with more of a long-term vision from the start (I-09). Mainly because the complexity could be kept lower with fewer dependent components, out of which some would be legacy components.

Further taking the value of other aspects into account, such as the previously mentioned market establishment, growing customer base, and speed, the value of accumulating early TD outweighed the cost for most companies interviewed.

To summarize, prototyping is an important part of start-up companies’ journey, through which companies can find and validate their market niche. Companies value agility and flexibility early on and consciously take on TD to facilitate this. The TD accumulated will in some cases slow down the implementation of feature requests, and will in most cases result in complete rewrites of the companies’ products when they start feeling confident about their way of doing things, or when an opportunity presents itself in the form of external funding. The opportunity cost of this approach is hard to estimate, given that no company has complete knowledge of their future path beforehand, leaving the possible outcome of building solid, scalable software in just the right way for the final product from the beginning, unknown. However, most companies interviewed have described their development process as the best one possible for them at the time. Given their knowledge going in, available resources, and skill set. This implies that the most common prototyping process is likely the way most start-ups will find success in implementing.

5.5 Pivoting

A clear majority of the companies interviewed were found to at some point have pivoted. The change of directions was prompted by different reasons. The interviewed companies commonly started out with an idea about a problem that could be solved for a specific user group and could be monetized through a certain business model. This initial idea was in most cases disproven after initial prototyping and market probing, prompting a change of direction (I-01, I-02, I-03, I-04, I-05, I-07). To what degree these companies had to change course was found to vary. Furthermore, the initial prototyping often led to the initial technology not being in line with the vision, customer segment, or scalability goal, often leading to full rewrites of the technology.

5.5.1 Business Influenced Pivoting

A common case of pivoting found was a change of target audience (I-01, I-02, I-04, I-05, I-07). Multiple companies had started out with the idea that the product would be marketed as a B2B or B2C product, which was later reversed to the opposite (I-01, I-05). Another case was companies that started out with a target audience of a specific industry in mind, which was later expanded to include customers from multiple industries (I-04, I-07). All these pivots commonly resulted in new technology and product design considerations that were not part of the initial plan or the initial product prototypes. The companies that pivoted from B2B to B2C and vice versa found themselves having to do major re-writes of their existing technology (I-01, I-05). These companies described the need for major rewrites as the result of major specification changes rather than TD artifacts of their existing technology. Increased modularity and flexibility were needed more than originally predicted. However, both companies seized the opportunity to rethink their system architecture and utilize leanings when rebuilding their product, ending up with a more scalable system with less TD for the future. Placing emphasis on doing things better the second time around.

"Since we switched to B2B, we kind of redid everything in a way that makes it easier for possible clients, the companies, to integrate with us ... We're not really developing any new features right now, most of the platform is as it was a few months ago. We're kind of trying to make everything so generic and scaleable that when somebody wants to

integrate our service at their own company, it might just take a few hours of work. So we're kind of working in that direction." (I-05)

Meanwhile, the companies that expanded their target audience to include more types of customers also had to rework large parts of the existing technology (I-04, I-07). These pivots were driven by an improved understanding of their markets and the market opportunity of their products. A larger market share could be gained and more revenue could be generated by broadening the product offerings rather than keeping a narrow focus. However, the expanded product scope turned out to work poorly with existing technologies built at these companies. Backend systems built scaled poorly with the new requirements, and the customizability of the frontend projects was a proven limiting factor of these pivots (I-04, I-07). These limiting factors could be argued to be artifacts of TD and resulted in friction pivoting effectively. Nevertheless, when reflecting on early technology decisions made which lead up to these TD artifacts, an interviewee emphasized the fact that decisions in start-ups are made with imperfect information about future events and where you as a company will be in the future. Decisions are made with the information available at the time (I-04). When asked about possible improvements to their planning process, and whether more time could have been spent initially to try and foresee such changes in product scope, the respondent answered that such exercise probably would not have benefited them:

"I think our approach was the completely right one. Even if we would have theorized and tried to analyze this pivot earlier, we wouldn't have arrived at the correct conclusions. Even if we would have spent more time on this, we would've ended up at the wrong end of the problem anyways. If that was the case, we would have likely had to make even bigger rewrites, because we would have built too much technology." (I-04)

A third type of pivot connected to changes in target audience was a company that started out with a B2C product, but later began packaging their product as a B2B product as well (I-02). This company was able to perform such a pivot without having to do major re-writes of existing technology. The reason was found to be that the company from the start had kept that market opportunity in mind and incorporated it into its long-term vision. Such long-term vision enabled them to plan for modularity and flexibility of their backend, actively working to keep TD low in parts of the system

central to such a pivot.

"I think we're in a relatively good position for this ... Even if it wasn't our initial intention, we have been conscious of this possibility and built our backend in a way that will support it, so it's been a consideration albeit not the primary one." (I-02)

Two of the companies interviewed had not yet made a pivot (I-03, I-06). These companies had started out with business plans and target audiences which were proven to be viable and profitable from a business perspective. These were also found to be the companies interviewed with the least turbulent product development journey out of all the interviewed companies. One of these companies had at the time of the interview not yet been forced to make any major re-writes or re-purposing of their software (I-06). The other company had done more product testing and adaptation to customer response but had been able to do so in an iterative fashion, without having to make any major changes of direction (I-03).

"We built things and largely they worked and then we improved on them and then changed things slightly, but then they worked as well. So we haven't had to pivot necessarily." (I-03)

To summarize, business-related pivoting often results in unavoidable TD. For many companies, as was previously described with prototyping, experimenting with the product, and establishing what product is attractive on the market is not something that could always be predicted, and will often need to be tested before any final decisions can be made. These decisions often emerged from the fact that the product didn't have a proven market, and a lot of iterating with different offers was needed, whereas when the product had been proven in other international markets, the concept could often be replicated and built upon to fit the domestic market.

5.5.2 Re-writes

As previously described, the early prototyping based on establishing a proof of concept and speed often led to technological decisions that were sub-optimal for the long-term vision. Especially when requirements, conditions, and the overall vision of a company change due to a pivot. These decisions implicitly led to entire products being TD, where companies at some point would have to completely scrap code and re-write it

from scratch in a different language, framework, or architecture, to adhere to the new product functionality, scalability, performance, user experience or any other specific needs (I-01, I-02, I-04, I-07, I-09). These technological re-writes also depended on in what stage they were performed, and how early companies were able to initiate the re-write, since as time goes on and work on the prototype continues instead of investing the time in performing the perhaps necessary technological shift, increasingly more TD is incurred (I-02, I-07, I-09).

”We have tried to scale the solution for a single user to the number of users we have. It’s not a good solution, we need a frontend that is customized for having multiple users at once. We are now in a position that if we build anything more on the current product, it will generate much technical debt.” (I-07)

Complete rewrites are often costly and require a large amount of resources with limited short-term gain (I-02, I-04, I-05, I-07, I-09). With restricted budgets, these changes may often be difficult to motivate management or other decision-makers, leading to TD increasing with interest (I-04, I-08). However, throughout the interviews, all start-ups that have performed necessary rewrites claim that it has brought great value in their subsequent business.

To summarize, pivots called for by market opportunities in customer segments other than initially targeted have been found to be common for start-ups. These pivots have in most cases resulted in new technological and product considerations and a shift in product requirements, something that often resulted in major technological re-writes. Whether the limitations of existing technology in all these cases can be attributed to TD is up for discussion. Two companies were mainly hindered by the flexibility and modularity of their existing software, while two companies were mainly hindered by the scalability and adaptability of previous technological decisions. In some cases, the velocity and how early these technological decisions could be implemented were restricted by budgets and customers, and by delaying the implementation of the technological change, TD was accrued. Even though this kind of pivot frequently required a large amount of effort and company resources, it was not found to discourage companies from making them eventually. The trade-off of cost and subsequent value was almost always tilted to the long-term value side when a certain threshold was reached. In addition, these pivots generally gave

companies the opportunity to re-evaluate their existing technologies and make better-informed decisions regarding the requirements and optimal design of their systems, often resulting in lower amounts of TD and more favorable conditions for future growth.

Chapter 6

Discussion

To be able to accomplish the aim of the study, to further understand the emergence of TD in start-ups, explore how TD is valued and managed at different stages, and shine a light on how it affects start-ups when they aim to scale up, a discussion based on the empiric results and previous analysis, in conjunction with existing literature is presented. The discussion aims to connect the empirical findings with existing literature, explore similarities, and discrepancies, and add further insight into the subject.

6.1 Emergence of Technical Debt

Technical debt is by and large unavoidable (Oliveira et al. 2015), and each start-up interviewed has experienced the effects of TD to a certain extent. As described in Steve McConnell's taxonomy of TD, there is unintentional- and intentionally accrued TD. However, as discussed by Martini et al. (2018), the optimal system architecture is often a trade-off between technological and business-related requirements. This calls for a somewhat more nuanced description of TD. The results of this study show that TD can be incurred unintentionally due to sloppiness, incurred intentionally for strategic reasons, and an intermediate origin of TD, where it is accrued due to an incomplete understanding of product and business requirements early in the development process. Further, multiple types of TD were found to emerge in start-up companies, including multiple types of TD found by Alves et al. (2016), such as documentation debt, people debt, and test debt. However, TD related to poorly written code was brought up more seldom than expected in the interviews. When it was, it was usually explained by

the work of short-term consultants or interns, which confirms the notion that Njima (2019) described surrounding alterations in teams and interns. This relative absence of code-related TD is interesting since immature or bad code was initially what TD referred to, and is one of the more prominent types of TD in the research reviewed. It could either be the case that the companies interviewed have written good enough code for it to rarely pose a major issue, or it could be that the negative effects of TD on the code level had not yet caught up with them. The most common type of TD brought up was architectural technical debt (ATD), revolving around poor system design, poor modularity, or sub-optimal technological choices. This gives merit to the claims of Soliman et al. (2021) regarding ATD, that it has the biggest impact on system quality, and that it is hard and costly to rectify after the product has been shipped. It might be the case that ATD should receive the most amount of attention from start-up companies since it is the type of TD that will show the worst symptoms first. However, these conclusions must be drawn with caution. As previously mentioned, the full spectrum of negative side effects of TD in other areas, such as code TD or people TD, might not yet have been discovered. As presented by Martini et al. (2015) and Martini et al. (2018), TD can remain hidden or unknown in software systems over time and become evident further down the line.

The case for agile software development presented in the theory section, that project or system requirements rarely are clear from day one but will change as development progresses (Allman 2012), was found to correlate with the reality of start-up companies in this study. Early-stage start-ups rarely had a clear vision of their future product and market that was proven true after initial market probing. Companies were often forced to do business-related pivots that had implications on the requirements of their technology. In an ideal scenario, companies would be able to foresee future technology requirements and make informed decisions from the beginning. However, the companies interviewed highlighted that creating elaborate product specifications or spending time and resources on planning wouldn't have been productive in the prototyping process. They expressed that they wanted to get going quickly and see what product features worked in the real world with real customers, before committing to perfecting them. Even though this strategy is valuable from a product and business perspective for companies, it did introduce TD for most companies interviewed, which in most cases had to be dealt with it later, usually through major re-writes. Some companies expressed that they did try to plan a few steps ahead, which reportedly did

save them some time further down the line and enabled them to respond favorably to market opportunities or pivots. In addition, the iterative and "make it up as we go" methodology of early-stage start-ups which could be argued to be the result of agile development practices, did in most cases result in lacking documentation, and large refactoring efforts were pushed on the future. These findings are in line with those of Oliviera et al. (2015) and Codabux et al. (2013), regarding the risk of shortsightedness associated with agile development methods. This study finds support for the claims of Njima and Demeyer (2019), that elements of more linear and standardized development processes to some degree could benefit start-ups and help them avoid TD in the early prototyping phase.

The intentional accumulation of early TD was a recurring theme of the interviews, with both positive and negative effects. The strategic and intentional use of shortcuts results in primarily ATD. The deliberate decisions that resulted in TD were based on the information and premises present at the time, much in line with what Verdecchia et al. (2021) argued. As previously discussed, the benefit of these deliberate shortcuts was the ability to quickly iterate and find the right product-market-fit, which is further validated through key findings in the literature that accepting TD at an early stage to develop prototypes may be beneficial (Cico et al. 2021). The negative implications included the need for switching technologies and system architecture further down the line, often time-consuming and resource-intensive efforts, leaving companies stuck with debt-laden systems until such projects were completed, affecting development speed and product performance.

Another source of TD that could be considered intentional, is the initial choice of technologies. In the interest of prototyping product features rapidly and validating product ideas, companies usually chose technologies that allowed them to do so and keep complexity low at an early stage. In most cases, this coincided with technologies the founding team had previous experience with. The results show that building prototypes, even with sub-optimal technologies for long-term scalability, was important to quickly test features and business ideas on the market. If the decision had been made to develop everything in an optimal technology from the start, the lack of competence would have impacted the development speed and could have resulted in missed market opportunities, just as Njima et al. (2019) discuss. As also found by Njima et al. (2019), adhering to the competencies of the company can be a source of TD. These decisions were mainly made constrained by skill set and previous experience,

focusing on short-term rewards rather than long-term vision.

A point that was seldom discussed to a further extent in the reviewed existing literature concerning TD was the impact of personality traits and team compositions. Besker et al. (2019), describe how the prioritization process of TD and adding new features was instigated via their conceptual model, and during the meetings, it was often decided what would be done by the people attending the meeting. The reasoning for this was not further discussed or motivated to a longer extent. However, the empirical results from this study show that the emergence and prioritization of TD were often affected by the personalities of team members and their considerations surrounding TD. Several of the interviewees contributed a large portion of their success by avoiding major implications because of TD to the thoroughness, awareness, and holistic mindset of lead technical managers such as CTOs and tech architects. It was also shown during the interviews that CTOs and tech leads that had previously experienced the pain of working with TD-riddled systems and had experience managing TD tried to steer away from accumulating to a further extent than their counterparts. The fact that there is a solid amount of expertise in the Stockholm area, with many successful start-ups originating from the area and venture capital being injected into the market, the management of TD within the start-ups may be of higher quality than a general global average.

Furthermore, team composition was also found to be an important factor in the avoidance of TD. As the results found that the team composition may in fact contribute more to a successful relationship with TD than planning would. De Almeida et al. (2018) argue that the tension between software development and business management contributes to the accumulation of TD, which could be an explanation for the importance of team compositions for the interviewed companies. In addition to this, as Njima et al. (2019) described, the stability of the teams is also a factor in TD, which was shown in the interviews as large amounts of accumulated TD in an area could be attributed to the use of consultants and interns.

Technical debt's effect on companies' ability to receive funding, and vice versa is a relationship that would be interesting to investigate further. This has not been the focal point of the study, and no external investors have been interviewed, restricting what conclusions can be drawn. However, the relationship between external funding and TD could be an interesting area for future research. A main concern of start-ups

is to test the validity of their product, quickly recruit new customers, and are likely to accumulate TD doing so. However, some companies participating in the study expressed that venture capital firms rarely consider the amount of TD in companies they invest in. The main decision criteria revolve around business and product-related aspects of the company, revenue, growth, and number of paying customers. These incentives could potentially result in a vicious cycle, ultimately creating worse conditions for companies to receive funding for every investment round they complete; Companies have to focus on gaining market shares, recruiting new customers, and adding features in order to receive funding. However, doing so will likely introduce TD, resulting in slower long-term development speed. When it's time for the next investment round, the company has accumulated TD and is in a worse position to continue developing its product, impeding its growth potential and making it harder to reach growth goals.

6.2 Management of Technical Debt

Besker et al. (2018), described that managing and avoiding TD significantly reduces the likelihood of technology pivoting in start-ups. However, the results show that whether or not the company would have managed the TD in an early prototype or product seldom influenced the decisions on switching to different technologies at a later point in time. The aspects that had the most influence on technology pivoting were often considered to be business criteria and factors related to scalability, such as capacity, but also to conditions such as customer satisfaction and popularity.

Furthermore, business pivots especially contribute to technological pivots. The results show that when the core market and business idea have been established from quick and iterative prototyping, the changes made in the business plan such as switching from B2C to B2B or altering the product offer, the new conditions may call for a transition to other technologies, such as programming languages or frameworks. The core vision proved in many cases to be helpful to manage and avoid TD during early and stabilizing phases. However, the core vision of the companies was in particular cases altered during the inception phase, leading to the vision change which needed to be addressed by technological pivots in many cases.

As Guo et al. (2016) argue, the initial cost of beginning to properly manage TD is large, but the recurring costs of managing it are implemented in the ongoing

development processes diminish swiftly. While not discussed in depth in the literature, the connection can be made that the costs associated with initializing a technological pivot could be considered to include the initial costs of beginning to manage TD. As the results showed, technological pivots give companies the opportunity to re-evaluate existing structures and make better-informed decisions, resulting in better management of TD along with increased scalability.

As discussed, the core vision has been shown during the interviews to have a major effect on how TD is managed for start-ups. With examples from the interviews such as the mantra "don't build to scale", where TD had become a large part of the day-to-day development, to visions that included them being completely self-funded and thereby avoiding TD to a large extent. The effects that a well-communicated and solid vision had on the TD in the development were tangible in many of the studied cases, even in the case where the business idea was "don't build to scale", a threshold was reached where the team needed to set a vision for scalability and avoidance of TD in order to progress. A strong vision aided the developers in making decisions that would not accumulate TD, as well as keeping them from developing features that would not be used and were not aligned with their vision, further avoiding TD. The vision also aided in making the correct structural choices, further minimizing TD and additional costs. The business vision's effect on the management of TD has not been identified as a point of study in the literature, however, the results show that there is value in establishing and discussing a company's vision at an early stage of a start-up.

This study encountered different attitudes and strategies toward documenting, tracking, and making TD visible. However, this study can confirm what Power (2013) stated, that tracking and managing TD is no trivial task. Managing TD requires a long-term perspective, planning, and commitment to follow procedures through, something that might be hard for start-ups to achieve effectively with limited resources. In order to work proactively with TD, it could be argued that a shared understanding of where TD resides in software systems, and what its potential consequences are, is an important prerequisite. Something also presented by Oliviera et al. (2015). As found in some of the interviews, simply hoping that these things will take care of themselves, is unlikely to provide companies with a clear understanding of where their TD lies, in a way that can easily be conveyed to people outside the development team. This might although not be a major issue for start-ups. As long as companies and their team are small, it can be possible to keep track of TD and foresee its future implications

without rigorous tracking methods. In small development teams, developers usually work closely together and each individual developer is familiar with most of the code base. Further, in the early stages of companies, communication chains are often short and product and business people are often involved in the development process. Based on the findings of this study, early-stage start-ups rarely suffer from the lack of TD tracking and visualization. The opposite might even be true, that they can spend time and resources more effectively elsewhere at an early stage. However, the fear expressed by some companies, and the danger inferred from speculative reasoning, is that insufficient documentation of TD can cause issues as companies transition into a scale-up phase and the number of employees grows. Onboarding new employees might become difficult due to undocumented TD and product and business people will have no way of considering TD in their decision-making and planning process. However, as described previously, as companies transition from an early prototyping stage to an early scale-up phase, the focus often shifts to incorporate more structured and documented development processes. Companies are often more reluctant to take on deliberate TD in product iterations that have already been rewritten from the initial prototyping stage. Whether this evolution of development processes at companies is enough to avoid negative consequences of early accumulated TD remains unanswered. Given the sample of companies interviewed in this study, it is not possible to draw any conclusions as to how the TD documenting, tracking, and visualization practices of early stage affect their long-term growth. Neither if the shift in prioritization and process as companies mature able to negate TD accumulated during the prototyping phase. These questions constitute interesting areas for future research.

6.3 Technical Debt and Scalability

Avgeriou et al. (2019), where the consensus of 33 practitioners from both academia and industry concluded that considering business value is critical when developing TDM practices, which is also argued in the IEEE International Conference on Software Maintenance and Evolution by de Almeida. During the interviews, it was found throughout the different cases that when business values were communicated with development teams, and vision was included in discussions, TD could be managed and avoided by the development teams. By including the developers in the vision, and having discussions on the implications of what they are developing in connection

with where they aim to be, TD could be avoided by adhering early to the scalability of the company, hence affecting the long-term business value. This would imply that for start-ups, the most suitable approach may not be to create TDM practices based on the business value per se but to include how the development may impact the business value and the vision through their actions, and include this as the backbone of the development practices. As was previously touched upon, the development processes were seldom structured during the early phases but became more established when the companies started to stabilize, hence there weren't standardized TDM practices in many of the companies at this stage. Hence, focusing on implementing a culture in the company that considers the scalability and business value in the development is found during the interviews to achieve success. However, some of the cases studied have not actually gone through a scale-up phase yet, but claim to be well-positioned for it. This must be taken into consideration when observing the results, and further studies must be performed before confirming this notion.

In addition, the team composition, skills, and personalities of team members were also found to impact the scalability of companies. Much in the same fashion as was just discussed, the personality traits of senior developers and their influence on the teams' approach to TD and scalability were found to be a large influence on the success of TDM and decision-making in regard to technological choices. The literature review acknowledged the team composition's impact via knowledge discrepancies between business management and development on the business value. Hence, succeeding with the team compositions and the communication of value and vision from management to development, and vice versa are considered in the results to be influential on the scalability and adaptability of the product.

From the companies interviewed, the transition from ad-hoc practices at an early stage to more structured work processes in a stabilization and scale-up phase was found to be impactful for the more mature companies, as well as something younger companies acknowledged would be helpful in their future practice. These more standardized processes also aid in managing the TD according to the results, where TD may swiftly grow in the "high clock-speed" environments (Banker et al. 2021). The TD accrued in this stage has a higher impact on the hindrance of scalability for start-ups (Besker et al. 2018).

The results found that one of the main hindrances of scalability was in fact TD.

Obstacles in the form of ATD such as sub-optimal systems and languages in use. These were often reconciled by major re-writes and system transfers, which gave the companies the opportunity to reflect on the scalability of the product and incorporate scalable factors in product development moving forward.

Chapter 7

Conclusion

Throughout the duration of this study, the aim has been to answer the research questions ” *How does TD emerge in Swedish tech start-ups?*”, *How is TD managed in Swedish start-ups?* and ” *How does the management of TD translate into the scale-up phase, and how do the start-ups ensure that the product is not hindered in its scalability by TD?*”. This has been done through a multiple-case study with relevant actors within the Swedish start-up scene.

In regards to the emergence of TD, the results have shown that the accumulation of TD arises in large during the swift and iterative inception phase, by developing prototypes swiftly with little concern regarding TD. This type of TD is however often more beneficial than costly to the start-ups because of the ability to establish a market presence and gather evidence on what demands there are for their products, thereby developing an optimal version. The TD accrued this way is often considered to be more easily adjusted, however, it is of utmost importance to address this TD early, before stabilizing and attempting to scale up because the TD accrued at this point quickly increases exponentially with interest, hindering any attempts at scaling up.

By establishing a solid vision, and communicating the vision along with the business value and the value of the product to the entire team, with the vision including scalability aspects, there were in many cases successful TD avoidance and management. By including a long-term vision and regarding each business and development decision with this in mind, the management and avoidance of TD often came implicitly, not needing to explicitly dedicate resources to handling TD.

The primary concern that was identified during the study, was the emergence and

effect of ATD. System and language choices based on comfort, knowledge, and development speed were often sub-optimal when scaling up, but beneficial at early stages. Furthermore, business pivots such as changing target groups also accumulate TD. This type of TD often led to complete re-writes and transferring to other systems and languages, leading to larger costs, the longer it took to initiate the re-write. The TD accrued by this was often the main hindrance to the company's ability to scale its product, however, the re-writes allowed the companies to re-evaluate business decisions and include scalability and TDM in their product and development plans.

The management of technical debt was seldom a question of routine checks in the early phases of the start-ups, and the tracking of TD was often limited to notes in the code, a few tickets, or the brain capacity of technical leads. However, timely re-writes and changing of languages and systems were often a solution to scalability issues, which also led to the elimination of TD, and an ability to re-evaluate and make better-informed decisions on how to manage TD from that clean slate. This often aided in establishing new practices of TDM, and the ability to replicate the product with scalability in mind and without TD.

7.1 Theoretical Implications

The study has found several aspects of the existing literature to be confirmed and disputed by the results and has also managed to identify new items of interest, where this study has gained new insights surrounding the managerial impact on TD.

One of the main findings in this study was in regards to the managerial aspects of TD accumulation and management, in terms of team composition and utilizing the skills within the company, and developing the company and the product with a long-term vision in mind. A key component in several of the cases was succeeding in gathering an optimal team for the cause, where some even stated that the team composition is more important than planning for avoiding TD, whereas the literature states that planning around TD is crucial. As Besker et al. (2019) state, is that the decision made around TD depends on what people attend the related meetings, but does not evolve the discussion around it. This study shows that the team, and the influencing personalities within, play a major role in how these start-ups have succeeded in managing TD. Thoroughness and attention to detail have been shown to be valuable traits in a company, but also

the sharing of the vision, and including all team members in how they want to work and establish a culture that generates holistic development and adds to the business value. It is obvious that a perfect team composition is wanted at any company, but the question remains, what is the perfect team composition in early start-ups with regard to how the development secures stability and avoids TD, and how would one negate the tensions in the cross-functional aspect that de Almeida et al. (2019), argues contributes to the accumulation of TD. Perhaps it is the right mix of personality traits within the team composition, and that is something that would need to be further investigated, as it might unlock new dimensions in how TD is treated.

Cico et al. (2021) found that accumulation of TD in a start-up phase could lead to technological pivots, where the entire tech stack gets replaced and the code base is rewritten (Cico et al. 2021b). Further, these technological pivots were found to be prompted by existing TD and realizations that a new tech stack could serve the company better and often led to more sustainable technology solutions (Cico et al. 2021b). However, the study of Cico et al. (2021) was conducted with a limited sample of companies, leading the authors to call for future research to validate their claims.

This study can confirm the findings of Cico et al. (2021) regarding start-up companies' habits of performing technological pivots. Our study finds evidence for technological pivots being the norm rather than an exception for start-ups as they emerge from an early prototyping phase. These technological pivots were found to mainly be prompted by TD accumulated deliberately during the prototyping phase to quickly validate business ideas, but also due to the expertise and skill set at the company, as well as companies refining their long-term vision and getting a better understanding of the technical requirements set by their product and their customers during the prototyping phase. In line with the observations of Cico et al. (2021), technological pivots often resulted in companies ending up with better systems that could scale more favorably, facilitating the development of new features and improving the performance of their products. Further, this study found that new and better-suited technological choices often coincided with the increased maturity of companies, putting more emphasis on avoiding TD and placing greater focus on long-term scalability. We believe our study can help validate the findings of Cico et al. (2021), as well as help explain the mechanisms of start-ups reaching these decisions. Early accrued TD related to sub-optimal technological choices such as programming languages or frameworks are in many cases conscious decisions. Companies benefit from starting

with technologies they are familiar with and that can help them iterate quickly. This type of TD, introduced for these reasons, has not been found to severely impact companies as they move closer to a scale-up phase. Especially since this type of TD, as previously mentioned, in most cases is permanently removed through technological pivots. However, it should be stated that these technological pivots in some cases were performed when companies reached their pain limit, where feature development had to be put on hold and large amounts of resources had to be diverted to the technological pivot.

To conclude, what the findings of our study imply, is that companies can benefit from anticipating technological pivots in their early prototyping and planning. Allowing themselves to spend less time at an early stage planning their infrastructure and deciding on the optimal technologies, since requirements will change along the way, and instead anticipate and plan for rewrites as the requirements become more clear through market probing, and a long-term vision starts to take shape.

7.2 Practical Implications

In terms of practical implications, the importance of properly considering the team composition and creating a balanced team could be shown to be crucial for the management of TD. The increased knowledge surrounding the managerial aspects surrounding TD and its mitigation will hopefully aid in start-ups minimizing the obstacles posed by TD. For founding members and employees, a larger weight on establishing a vision and creating the right circumstances for development should be added, and identifying the optimal personalities for the team composition. Team members with a holistic and business perspective are more prone to solve problems with a long-term approach.

Further, given the history of the companies interviewed in this study, in conjunction with findings from previous studies, technological pivots are to be expected not long after the initial prototype has been developed. This will likely be caused by early accumulated architectural technical debt. This can in part be counteracted by spending more time envisioning future scenarios and planning a flexible system architecture, built with programming languages and technologies that will scale. However, as found in this study, product requirements can be hard to foresee and are likely to change after initial market probing. These insights are free for practitioners to evaluate given their

situations and use cases. However, we present the case for a pragmatic approach to this problem. Practitioners can benefit from early realizing that prototypes can fill a purpose other than constituting the technological base of a long-lived software system. Tearing up early technology and starting over should perhaps not only be regarded as sunken costs, and the time it takes to reimplement the product in new technologies with different architectures should perhaps not be regarded as "wasted time". The benefits of moving quickly early and unveiling unknown unknowns right away can be valuable in finding the right product-market fit with minimal resources.

7.3 Limitations

A concern when it comes to the semi-structured interviews is that it is difficult to evaluate whether the topics have been sufficiently covered (Kelly 2010). Furthermore, the interviews may be influenced by the participants' moods, biases, and ideologies (Kelly 2010). This needs to be taken into regard when evaluating how to establish the validity of the study.

One of the main limitations, which the authors must be humble about, is that the interviewees were primarily in the early stage of their start-up journey. This implies that although the interviewees have seen success so far in their companies, their way of handling TD is not guaranteed to bridge the gap with more mature companies. The study's scope has however focused on the early stages of start-ups, and the possible complications of a much later stage are therefore not included in the study, but one may wish to further examine how mature companies successfully navigated TD in their start-up phase.

Survivor bias is a concept that is present within the study. The authors were only able to gather data on companies that have succeeded in their products and are currently in an ongoing process of managing their TD, where some are more successful than others. The study has not been able to include start-ups that have failed with managing their TD and have undergone bankruptcy and closed down their companies. This must be acknowledged and considered during the study.

7.4 Future Work

The study has found promising results considering how the team composition, personality traits, vision, and inclusion of business values in the development process affect the scalability of the start-ups in Swedish tech start-ups with regard to TD. There were few cases in the study where the company had accomplished a scale-up phase, however, future studies which include more mature software companies that have experienced a successful scale-up phase are needed to validate the results found.

Further, an interesting area for future research is the interdependence of TD accumulation in start-ups, and the ability to receive funding. The rationale is that investors are interested in portfolio companies' revenue, market share, product functionality, and growth. Attempting to reach such goals could potentially create incentives for start-ups to take shortcuts, introducing TD in their software, which in turn reduces the mid-term or long-term development speed of future development, inhibiting the ability to reach growth targets in the future. In addition, the notion hinted in this study is that venture capital firms give little to no attention to the TD of companies they invest in, further creating incentives for start-ups to take shortcuts that could prove costly in the future. Furthermore, the use of consultants in this context, where funds are increased and, as the results showed, more TD is accrued with the use of consultants but quick results are made, is worth investigating in future research.

An interesting point of discussion could be made surrounding the character of TD in relation to business-related pivoting. The character of this type of TD may be more related to managerial decisions, in contrast to a technical level, even though the costs associated with it are related to technological development, this type of TD may be a separate classification on its own. This constitutes an interesting topic for further research, on what the consequences of this angle of thought would be, and how this could aid the successful navigation of TD for start-ups.

Finally, this study has been conducted on early-phase start-ups, start-ups approaching a scale-up phase, as well as start-ups currently experiencing a scale-up phase. However, no mature company where the long-term effects of TD have become evident has been interviewed. It would be interesting to investigate how the unstructured and undocumented TDM common in start-ups, translates into long-term effects of companies. Also, it would be interesting to investigate how start-up companies shift

towards more structure and process once a clear business case has been proven to affect the long-term TDM of companies. It is unclear whether this introduction of better processes should ideally be considered earlier to avoid the negative consequences of TD accumulation. This is also an interesting area for future studies.

Bibliography

- [1] Allman, Eric. “Managing Technical Debt: Shortcuts that save money and time today can cost you down the road.” In: *Queue* 10 (2012). <https://doi.org/10.1145/2168796.2168798>.
- [2] Almeida, Rodrigo Reboucas de et al. “Aligning Technical Debt Prioritization with Business Objectives: A Multiple-Case Study”. In: *PROCEEDINGS 2018 IEEE INTERNATIONAL CONFERENCE ON SOFTWARE MAINTENANCE AND EVOLUTION (ICSME)*. Proceedings-IEEE International Conference on Software Maintenance. ISSN: 1063-6773. IEEE; IEEE Comp Soc, 2018. <https://doi.org/10.1109/ICSME.2018.00075>.
- [3] Alves, Nicolli S. R. et al. “Identification and management of technical debt: A systematic mapping study”. In: *INFORMATION AND SOFTWARE TECHNOLOGY* 70 (2016). <https://doi.org/10.1016/j.infsof.2015.10.008>.
- [4] Apa, Cecilia et al. “The Perception and Management of Technical Debt in Software Startups”. In: *Fundamentals of Software Startups: Essential Engineering and Business Aspects*. Ed. by Anh Nguyen-Duc et al. Cham: Springer International Publishing, 2020. https://doi.org/10.1007/978-3-030-35983-6_4.
- [5] Avgeriou, Paris et al. “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)”. In: (2016). In collab. with Marc Herbstritt. Artwork Size: 29 pages Medium: application/pdf Publisher: Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik GmbH, Wadern/Saarbruecken, Germany. <https://doi.org/10.4230/DAGREP.6.4.110>.

- [6] Baham, Corey. "Exploring the Relationship between Perceptions of Agile Software Development and Technical Debt". In: *AMCIS 2017 Proceedings* (2017).
- [7] Balawi, Ayman and Ayoub, Asad. "Assessing the entrepreneurial ecosystem of Sweden: a comparative study with Finland and Norway using Global Entrepreneurship Index". In: *Journal of Business and Socio-economic Development* 2 (2022). Publisher: Emerald Publishing Limited. <https://doi.org/10.1108/JBSED-12-2021-0165>.
- [8] Banker, Rajiv, Liang, Yi, and Ramasubbu, Narayan. "Technical Debt and Firm Performance". In: *MANAGEMENT SCIENCE* 67 (2021). <https://doi.org/10.1287/mnsc.2019.3542>.
- [9] Baxter, Pamela and Jack, Susan. "Qualitative Case Study Methodology: Study Design and Implementation for Novice Researchers". In: *The Qualitative Report* 13 (2008). <https://doi.org/10.46743/2160-3715/2008.1573>.
- [10] Becker, Paul et al. *Refactoring: Improving the Design of Existing Code*. Google-Books-ID: UTgFCAAAQBAJ. Addison-Wesley Professional, 1999. 461 pp.
- [11] Bell, Emma and Bryman, Alan. "The Ethics of Management Research: An Exploratory Content Analysis". In: *British Journal of Management* 18 (2007). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8551.2006.00487.x>. <https://doi.org/10.1111/j.1467-8551.2006.00487.x>.
- [12] Besker, Terese, Martini, Antonio, and Bosch, Jan. "Technical Debt Triage in Backlog Management". In: *2019 IEEE/ACM INTERNATIONAL CONFERENCE ON TECHNICAL DEBT (TECHDEBT 2019)*. IEEE; Assoc Comp Machinery; IEEE Comp Soc, 2019. <https://doi.org/10.1109/TechDebt.2019.00010>.
- [13] Besker, Terese et al. "Embracing Technical Debt, from a Startup Company Perspective". In: Book Title: *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)* ISSN: 1063-6773. IEEE, 2018. <https://doi.org/10.1109/ICSME.2018.00051>.
- [14] Besker, Terese et al. "The influence of Technical Debt on software developer morale". In: *JOURNAL OF SYSTEMS AND SOFTWARE* 167 (2020). <https://doi.org/10.1016/j.jss.2020.110586>.

- [15] Bomfim Jr., Marcelo M. and Santos, Viviane A. "Strategies for Reducing Technical Debt in Agile Teams". In: *AGILE METHODS, WBMA 2016*. Ed. by TS DaSilva et al. Vol. 680. Communications in Computer and Information Science. ISSN: 1865-0929. Araucaria Fdn; Natl Council Sci & Technol Dev; Fed Univ Parana; Fed Univ of Sao Paulo; Pontifical Univ Catholic Rio Grande Sul, 2017. https://doi.org/10.1007/978-3-319-55907-0_6.
- [16] Braun, Virginia and Clarke, Victoria. "Using thematic analysis in psychology". In: *Qualitative Research in Psychology* 3 (2006). <https://doi.org/10.1191/1478088706qp063oa>.
- [17] Cavallo, Angelo, Ghezzi, Antonio, and Balocco, Raffaello. "Entrepreneurial ecosystem research: present debates and future directions". In: *International Entrepreneurship and Management Journal* 15 (2019). <https://doi.org/10.1007/s11365-018-0526-3>.
- [18] Cavallo, Angelo et al. "Fostering digital entrepreneurship from startup to scaleup: The role of venture capital funds and angel groups". In: *Technological Forecasting and Social Change* 145 (2019). <https://doi.org/10.1016/j.techfore.2019.04.022>.
- [19] Cico, Orges et al. "Startups Transitioning from Early to Growth Phase - A Pilot Study of Technical Debt Perception". In: *Software Business*. Ed. by Eriks Klotins and Krzysztof Wnuk. Lecture Notes in Business Information Processing. Cham: Springer International Publishing, 2021. https://doi.org/10.1007/978-3-030-67292-8_8.
- [20] Cico, Orges et al. "Toward a Technical Debt Relationship with the Pivoting of Growth Phase Startups". In: *Product-Focused Software Process Improvement: 22nd International Conference, PROFES 2021, Turin, Italy, November 26, 2021, Proceedings*. Berlin, Heidelberg: Springer-Verlag, 2021. https://doi.org/10.1007/978-3-030-91452-3_18.
- [21] Codabux, Zadia and Williams, Byron. "Managing technical debt: An industrial case study". In: *2013 4th International Workshop on Managing Technical Debt (MTD)*. 2013 4th International Workshop on Managing Technical Debt (MTD). 2013. <https://doi.org/10.1109/MTD.2013.6608672>.
- [22] Creswell, John. "Research Design: Qualitative, Quantitative, and Mixed-Method Approaches". In: 2009.

- [23] Cunningham, Ward. *The WyCash portfolio management system*. 1992.
- [24] Dalal, Vishal et al. *Tech debt: Reclaiming tech equity* | McKinsey.
- [25] Ernst, Neil A. et al. "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt". In: *2015 10TH JOINT MEETING OF THE EUROPEAN SOFTWARE ENGINEERING CONFERENCE AND THE ACM SIGSOFT SYMPOSIUM ON THE FOUNDATIONS OF SOFTWARE ENGINEERING (ESEC/FSE 2015) PROCEEDINGS*. Ed. by E DiNitto, M Harman, and P Heymans. ACM SIGSOFT; ABB; Microsoft Res; Fujitsu; CEFRIEL; Google; BTO; ITA STQB & Engn, 2015. <https://doi.org/10.1145/2786805.2786848>.
- [26] Fatemi, Falon. *Technical Debt: The Silent Company Killer*. Forbes. Section: Hollywood & Entertainment.
- [27] Fungprasertkul, Suwichak and Bahsoon, Rami. "Managing Technical Debt Under Uncertainty". In: *IEEE SOFTWARE* 38 (2021). <https://doi.org/10.1109/MS.2021.3101945>.
- [28] Guo, Yuepu, Spinola, Rodrigo Oliveira, and Seaman, Carolyn. "Exploring the costs of technical debt management - a case study". In: *EMPIRICAL SOFTWARE ENGINEERING* 21 (2016). <https://doi.org/10.1007/s10664-014-9351-7>.
- [29] Hammarberg, Anna. *The land of unicorns*. Business Sweden. 2020.
- [30] Holvitie, Johannes et al. "Technical debt and agile software development practices and processes: An industry practitioner survey". In: *INFORMATION AND SOFTWARE TECHNOLOGY* 96 (2018). <https://doi.org/10.1016/j.infsof.2017.11.015>.
- [31] Institute, Swedish. *Sweden Tech Ecosystem: Report 2021*. Swedish Institute, 2022.
- [32] Kallio, Hanna et al. "Systematic methodological review: developing a framework for a qualitative semi-structured interview guide". In: *Journal of Advanced Nursing* 72 (2016). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/jan.13031>. <https://doi.org/10.1111/jan.13031>.

- [33] Kelly, Susan E. “Qualitative Interviewing Techniques and Styles”. In: Bourgeault, Ivy, Dingwall, Robert, and De Vries, Raymond. *The SAGE Handbook of Qualitative Methods in Health Research*. 1 Oliver’s Yard, 55 City Road, London EC1Y 1SP United Kingdom: SAGE Publications Ltd, 2010. <https://doi.org/10.4135/9781446268247.n17>.
- [34] Kitto, Simon, Chesters, Janice, and Grbich, Carol. “Quality in qualitative research”. In: *The Medical journal of Australia* 188 (2008). <https://doi.org/10.5694/j.1326-5377.2008.tb01595.x>.
- [35] Kruchten, Philippe. “Strategic Management of Technical Debt”. In: *2012 12TH INTERNATIONAL CONFERENCE ON QUALITY SOFTWARE (QSIC)*. Ed. by A Tang and H Muccini. International Conference on Quality Software. ISSN: 1550-6002. IEEE Comp Soc; IEEE Reliabil Soc; NW Polytechn Univ; Univ Hong Kong; Univ Aquila, 2012. <https://doi.org/10.1109/QSIC.2012.17>.
- [36] Li, Zengyang, Avgeriou, Paris, and Liang, Peng. “A systematic mapping study on technical debt and its management”. In: *JOURNAL OF SYSTEMS AND SOFTWARE* 101 (2015). <https://doi.org/10.1016/j.jss.2014.12.027>.
- [37] Martini, Antonio, Besker, Terese, and Bosch, Jan. “Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations”. In: *Science of Computer Programming* 163 (2018). <https://doi.org/10.1016/j.scico.2018.03.007>.
- [38] Martini, Antonio, Bosch, Jan, and Chaudron, Michel. “Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study”. In: *Information and Software Technology* 67 (2015). <https://doi.org/10.1016/j.infsof.2015.07.005>.
- [39] Martini, Antonio, Sikander, Erik, and Madlani, Niel. “A semi-automated framework for the identification and estimation of Architectural Technical Debt: A comparative case-study on the modularization of a software component”. In: *INFORMATION AND SOFTWARE TECHNOLOGY* 93 (2018). <https://doi.org/10.1016/j.infsof.2017.08.005>.
- [40] Nguyen-Duc, Anh, Seppänen, Pertti, and Abrahamsson, Pekka. “Hunter-gatherer cycle: a conceptual model of the evolution of software startups”. In: *Proceedings of the 2015 International Conference on Software and*

- System Process*. ICSSP 2015. New York, NY, USA: Association for Computing Machinery, 2015. <https://doi.org/10.1145/2785592.2795368>.
- [41] Njima, Mercy. “Architecting for scale: the case for systematic software reuse in managing technical debt in start-ups”. In: *Proceedings of the 13th European Conference on Software Architecture - Volume 2*. ECSA '19. New York, NY, USA: Association for Computing Machinery, 2019. <https://doi.org/10.1145/3344948.3344950>.
- [42] Njima, Mercy and Demeyer, Serge. “Value-Based Technical Debt Management: An Exploratory Case Study in Start-Ups and Scale-Ups”. In: *PROCEEDINGS OF THE 2ND ACM SIGSOFT INTERNATIONAL WORKSHOP ON SOFTWARE-INTENSIVE BUSINESS: START-UPS, PLATFORMS, AND ECOSYSTEMS (IWSIB '19)*. Ed. by K Smolander et al. Assoc Comp Machinery; ACM SIGSOFT, 2019. <https://doi.org/10.1145/3340481.3342739>.
- [43] Oliveira, Frederico, Goldman, Alfredo, and Santos, Viviane. “Managing Technical Debt in Software Projects Using Scrum: An Action Research”. In: *2015 AGILE CONFERENCE*. Agile Coaching Inst; AGILE; Atlassian; Axosoft; Blue Agil; Zooz Aien Hamilton; Temenos; Capital One; Elect Cloud; Every Voice Engaged; InfoQ; Cutter Consortium; LitheSpeed; Leankit; Esther Derby; Workfront; Hansoft; IBM; Agile; Mindtree; Inteliware; ICON; Inflectra SpiraTeam; InfirmIT.com; JelBRAINS; ScrumStudy; Methods & Tools; Lockneed Martin; Spartez; Planningpoker; HP; PHI; CA; HALLT; Scrum.Org; Scrum Alliance; SD Times; Skytap; Mingle; SolutionsIQ; NEOTYS; IC Agile; CenturyLink; TEMPO; UST; Scott Ambler Assoc; TRICENTIS; VersionOne; Eliassen Grp; Agile Craft; Z PHYR; Swift Kanban; Cardboard; TEK Syst global Serv; Scaled Agile; ASPE; InfoZen; Shippable; Pendo; ProjectCooks; Dovel Icclevol, 2015. <https://doi.org/10.1109/Agile.2015.7>.
- [44] Power, Ken. “Understanding the Impact of Technical Debt on the Capacity and Velocity of Teams and Organizations”. In: *2013 4TH INTERNATIONAL WORKSHOP ON MANAGING TECHNICAL DEBT (MTD)*. 2013.
- [45] Publishing, OECD. *Start-ups and Innovative Entrepreneurship*.
- [46] Rios, Nicolli et al. “Causes and Effects of the Presence of Technical Debt in Agile Software Projects”. In: (2019). Accepted: 2019-10-31T17:29:16Z. <https://doi.org/10.13016/m2ez7f-mfj6>.

- [47] Rubin, Kenneth S. *Essential Scrum: A Practical Guide to the Most Popular Agile Process*. Google-Books-ID: 3vGEcOfCkdwC. Addison-Wesley, 2012. 501 pp.
- [48] Saunders, Mark. *Research methods for business students*. In collab. with Adrian Thornhill and Philip Lewis. Seventh edition.. New York, New York State: Pearson Education, 2015. xxvi+741.
- [49] Shah, Sonali K. and Corley, Kevin G. “Building Better Theory by Bridging the Quantitative–Qualitative Divide*”. In: *Journal of Management Studies* 43 (2006). _eprint: <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-6486.2006.00662.x>. <https://doi.org/10.1111/j.1467-6486.2006.00662.x>.
- [50] Soliman, Mohamed, Avgeriou, Paris, and Li, Yikun. “Architectural design decisions that incur technical debt - An industrial case study”. In: *INFORMATION AND SOFTWARE TECHNOLOGY* 139 (2021). <https://doi.org/10.1016/j.infsof.2021.106669>.
- [51] Stam, Erik and Ven, Andrew van de. “Entrepreneurial ecosystem elements”. In: *Small Business Economics* 56 (2021). <https://doi.org/10.1007/s11187-019-00270-6>.
- [52] StartupBlink. *Global Startup Ecosystem Index 2022*. 2022.
- [53] Stockholms Handelskammare. *Därför är riskkapitalbolagen avgörande för Stockholms startupscen | Stockholms handelskammare*. Stockholms Handelskammare. 2023.
- [54] Tornhill, Adam and Borg, Markus. *Code Red: The Business Impact of Code Quality – A Quantitative Study of 39 Proprietary Production Codebases*. 2022.
- [55] Verdecchia, Roberto et al. “Building and evaluating a theory of architectural technical debt in software-intensive systems”. In: *JOURNAL OF SYSTEMS AND SOFTWARE* 176 (2021). <https://doi.org/10.1016/j.jss.2021.110925>.
- [56] Westlund, Hans, Olsson, Amy Rader, and Larsson, Johan P. “Economic Entrepreneurship, Startups and Their Effects on Local Development: The Case of Sweden”. In: (2011).
- [57] Wiese, Marion, Riebisch, Matthias, and Schwarze, Julian. “Preventing Technical Debt by Technical Debt Aware Project Management”. In: *2021 IEEE/ACM INTERNATIONAL CONFERENCE ON TECHNICAL DEBT (TECHDEBT 2021)*.

- IEEE; Assoc Comp Machinery; IEEE Comp Soc, 2021. <https://doi.org/10.1109/TechDebt52882.2021.00018>.
- [58] Yin, Robert K. *Case Study Research: Design and Methods*. Google-Books-ID: FzawIAdilHkC. SAGE, 2009. 241 pp.
- [59] Yli-Huumo, Jesse, Maglyas, Andrey, and Smolander, Kari. "How do software development teams manage technical debt? - An empirical study". In: *JOURNAL OF SYSTEMS AND SOFTWARE* 120 (2016). <https://doi.org/10.1016/j.jss.2016.05.018>.
- [60] Yli-Huumo, Jesse, Maglyas, Andrey, and Smolander, Kari. "The Sources and Approaches to Management of Technical Debt: A Case Study of Two Product Lines in a Middle-Size Finnish Software Company". In: *PRODUCT-FOCUSED SOFTWARE PROCESS IMPROVEMENT, PROFES 2014*. Ed. by A Jedlitschka et al. Vol. 8892. Lecture Notes in Computer Science. ISSN: 0302-9743. 2014.

Appendix - Contents

A Appendix

86

Appendix A

Appendix

Interview template

Quick introduction of us and our study. (5 minutes)

Initial - Quickfire questions (5 min)

Place your company on this Technical Readiness Level (TLR) - Scale

- How familiar are you with the term Technical Debt personally? 1-10

High Familiarity with Technical Debt

- How much do you know about Technical Debt within your company? 1-10
- To what extent do you actively work with technical debt today? 1-10
- How big a problem is it for you at the moment? 1-10

Low Familiarity

- Do you feel that the development of your product has been hindered due to previous technical decisions? 1-10
- How much of the company sits on the knowledge-base around the development of the product? 1-10
- How big of a problem is this type of obstacle for you? 1-10

We want to "loosen up" the interviewee with these initial questions, and gain knowledge surrounding the interviewees awareness and knowledge of technical debt

within their company, in order to adapt the following questions to their knowledge level.

Background / context - Shorter Questions (10 min)

- Question 1 - Can you briefly tell us about the emergence of the idea behind the product?
- Question 2 - What was your view of the market while you were building the product? Competition, rate of change, etc.
- Question 3 - How many were you that developed the technology behind the product(s)?
- Question 4 - What was your financing situation?
- Question 5 - What was the product development timeline? Did you experience pressure, or did it develop slowly over time?
- Question 6 - What are your current products?
- Question 7 - How are you further developing the product, both technically as well as meeting market demands.
- Question 8 - Are you experiencing problems with maintaining or further developing the product?

Objectives with questions 1-8: Understand the company's conditions, environment and positioning. With the help of this, be able to deepen the analysis around the origin and management of technical debt in startups. Enable comparisons and parallels between different companies in the study. Have companies that pivoted a lot incurred more technical debt? Have companies that have been exposed to time pressure/high competition incurred more technical debt than companies that had more time on their hands? Does team size and funding play a role in the amount of technical debt? By being able to answer these questions, we have the opportunity to draw conclusions about where, when and why technical debt mainly occurs in startups, as well as being able to generate strategies to best prevent/take advantage of technical debt given a company's conditions.

Open questions (40 min)

Question 1 Can you tell us how the product evolved and transformed during its development.

Probing/follow up

- Is the product today similar to the one you had the ambition to create?
- How did the functionality/purpose of the product change over time? Was functionality built with one purpose, which then had to fill another?
- What changes in direction have you taken?
- Have you built systems/applications with a certain technology/architecture, which then had to be moved to another?

The aim of this question is to understand where technical debt has occurred during its development by actions such as pivoting, and understand the motivation behind the pivoting and what the results were. What were the tradeoffs of pivoting.

Question 2 - Can you tell us about any decisions you've made, such as changes in direction, pushed deadlines, investor-based decisions, that then created obstacles in other areas?

Here we want to find out if they have made strategic decisions that, consciously or unconsciously, affected the company's technical debt.

Question 3 - Work Process - Can you describe your development process? How is the team/teams organized?

Probing/follow up

- What has been the pros/cons with this way of organising and working?
- Has it evolved during the lifetime of the company? What has been the pros and cons with the different variants?
- Do you set aside time for refactoring/maintenance regularly, or is it done when a pain threshold is reached?

Here we want to link to the agile way of working. Gain an understanding of how start-ups work in projects, and find out what they think works well/poorly. We also want to find out about how technical debt is connected to the way of working.

Question 4 - How do you manage technical debt today?

Probing/follow up

- Is it handled when it arises or do you work preventively to prevent the occurrence of technical debt/technical obstacles?
- Do you use/have you used any tool to keep track of flaws/risks in your codebase?
- What do you think works well/poorly with the way you handle the emergence of technical debt today?
- Have you handled it in any other way historically? Advantages/disadvantages in such cases.

Here we want to find out how the management looks today, what works well or poorly with this. Is the company aware of the technical debt in the products or is it handled like any other problem.

Question 5 - Can you tell us about what your plans look like for the growth phase, alternatively what your plan looked like during the growth phase and how the implementation of this went?

Probing/follow up

- What obstacles do you think/did you think you would encounter?
- What do you think will go well/went well/better than expected?
- How do you ensure that the product is scalable?

Here we want understand what the growth phase looked like for the company. Alternatively, how they see that they should implement and scale their product in a possible growth phase. What obstacles do they think they might encounter during such a phase? What do the plans look like to make the product scalable, alternatively how did the plans go during the growth phase, what obstacles were they prepared/not prepared for, what went better than they thought.