Security Debt in Practice

A Qualitative Case Study

Maren Maritsdatter Kruke

Thesis submitted for the degree of
Master in Informatics: Programming and System Architecture
60 credits

Department of Informatics
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Security Debt in Practice

A Qualitative Case Study

Maren Maritsdatter Kruke

Abstract

Context
Implementing the required security at all the needed layers of the software product is important as malicious actors find that there is a greater potential gain in cyber-attacks. It is therefore important to be aware of the solutions in the software systems that does not meet the desired security goal. Properly managing these sub-optimal solutions is essential for keeping them under control.

Objective
The goal of this study is to provide further insight into the security debt phenomenon by producing a security debt definition, a way to manage this security debt, and find the relation between security debt and technical debt (TD). These three aspects are looked into in order to answer the research problem: **what is security debt, how is it managed in practice, and what is its relation to technical debt?**

Method
An exploratory case study was conducted. I collected data using semi-structured interviews and a brief document study. A total of 26 software practitioners from one company were interviewed. Their answers have been analyzed and presented.

Result
The following security debt definition is proposed on the basis of the respondents answers: **security debt is a set of design or implementation solutions that hinder or has the potential to hinder the achievement of a system's optimal/desired/required security goal**. Security vulnerabilities have been shown to be a part of security debt to a varying degree. Two figures are proposed to show their relation. The technical debt manage-

ment process have been described by respondents to also be relevant for the management of security debt. It has also been pointed out that security debt have a higher priority than technical debt. The three additional approaches for managing security debt mentioned by the respondents are threat modelling, bug bounty program, and security testing. The governing factors for the backlog prioritization is communication, the use of labels, and adding a severity/priority score. Security knowledge have been described to be important in the management of security debt. Finally, evidence of a relation between security debt and technical debt have been presented, visualized by two figures.

Conclusion
The security debt definition proposed in this study differs from other attempts to define security debt through the focus on the potential to hinder the achievement of a system's security goal. The definition provides the basis for describing the difference between security debt and security vulnerabilities; security vulnerabilities that do not have existing solutions cannot be security debt as there is no room for improvement. The security debt management can benefit from having approaches that are more security oriented as security debt combines technical debt and security. The main factors for security debt prioritization is communication, labels, and severity/priority. A lack in the needed security knowledge can contribute to the accumulation of security debt. Evidence have been found to support security debt being a part of the technical debt landscape. It has been further observed that there is a relation between architectural technical debt (ATD) and security debt. Having different approaches for the different types of technical debt can be beneficial as it supports the individual needs of the specific types of debt. Thus, security oriented approaches can benefit the management of security debt.

Acknowledgements

This thesis marks the end of the two-year master's program Informatics: Programming and System Architecture. It has been both an educational and challenging journey for me. I had a lot of support and this thesis would not have been possible without many awesome people!

Firstly, I would like to thank my supervisor Antonio Martini (University of Oslo) and co-supervisor Daniela Soares Cruzes (Norwegian University of Science and Technology) for the support and guidance, and not to mention patience! Additionally, I would like to thank my company contact and all my respondents for their contributions in making this study possible.

To my family and friends who supported me on this journey, thank you!

Maren Maritsdatter Kruke

Oslo, May 2022

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Our society is increasingly becoming more digitalized and computers are connecting most of the critical infrastructure and the functions we rely on in our daily lives. We rely on these software systems to be engineered so that they continue to "*function correctly under malicious attacks*" (McGraw, 2004, p. 80), that they are of the required quality, or security (ISO/IEC, 2011). Security is to a greater extent an important part of the development of software systems due to malicious actors seeing greater potential gain in cyber-attacks (Ardi et al., 2007). Even though security is becoming a more important concern, security is in many cases added as an afterthought because software developers are rather more focused on functionality (Maymi & Harris, 2019, p. 1083). There is a desire for short time-to-market as this allows a company to get the systems to the end-users earlier (Kruchten et al., 2013). The result is an increase in technical debt and a reduction of the software quality (Lindgren et al., 2008) (e.g. security). Technical debt is a fairly new concept that has been both refined and expanded since its introduction (Kruchten et al., 2012) by Cunningham, 1992. Avgeriou et al., 2016 defined technical debt as:

"*a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability*" (Avgeriou et al., 2016, p. 112).

Technical debt is detrimental to software systems (Lim et al., 2012) and is, according to Fowler, something that all development teams accumulate, even the most experienced teams (Fowler, 2009), in their practice. Managing the technical debt is central for keeping it under control. Technical debt can be an investment (e.g. for shorter time-to-market) as long as the development teams working on the software systems are aware of the debt that is being accumulated. Additionally, the increased risk to the system due to the technical debt should also be something the teams considers and tracks (Kruchten et al., 2012) in their practice.

The metaphor of technical debt encompasses various types of debt, but the definition does not cover all the important topics that needs studying (Avgeriou et al., 2016). Rios et al., 2018 have identified 15 types of technical debt; design debt, code debt, architectural technical debt, test debt, documentation debt, defect debt, infrastructure debt, requirements debt, people debt, build debt, process debt, automation test debt, usability debt, service debt, and versioning debt. Not only is the quality attributes maintainability and evolvability, as explained by Avgeriou et al., 2016 in their technical debt definition, impacted by technical debt, but Li et al., 2015 and Tom et al., 2013 pointed out that other quality attributes such as usability, reliability, security, etc. may also be compromised due to technical debt. Therefore, further research on the affected quality attributes in the form of debt is required, which brings us to security debt.

Security debt is understood as "*the intersection of security aspects with technical debt*" (Martinez et al., 2021, p. 1). Security debt is a term that has been attracting increasing attention by practitioners and researchers. As a result, security debt has been described in research throughout the last few years. However, a formal definition has not been offered. Publications, such as Martinez et al., 2021; Rindell et al., 2019; Silva et al., 2016 and Rindell and Holvitie, 2019 discussed security debt definitions. In addition, Siavvas et al., 2020 pointed out that Izurieta et al., 2018 and Izurieta and Prouty, 2019 provided initial attempts to define security debt. These publications were used as data analysis entries when Martinez et al., 2021 aimed to propose a definition of security debt, map its main characteristics, find out how security debt is managed throughout the life-cycle of a software product, and identify the security debt items in the current literature.

The findings above indicate the need for further research on the topic of security debt and the relation to technical debt. A closer look at security debt in connection with technical debt can help determine the extent to which security debt is harmful and how this debt should be handled. Thus, the objective of this thesis is to bridge this gap. This is accomplished by analyzing software practitioners' views on security debt and how they think it should be handled.

1.1 Research problem

Based on the literature presented above, the defined research problem is as follows:

What is security debt, how is it managed in practice, and what is its relation to technical debt?

1.2 Research questions

RQ1 How is security debt defined?

RQ1.1 What is the difference between security debt and security vulnerabilities?

RQ2 How is security debt managed?

RQ2.1 How is security debt prioritized compared to other types of technical debt, feature development, bug fixing, etc.?

RQ2.2 What role does security knowledge play in the different activities of security debt management?

RQ3 What is the relation between security debt and technical debt?

Listed above are the six questions created for answering the research problem. The three main research questions (RQ1, RQ2, and RQ3) aim to find out how security debt can be defined, how it can be managed, and its relation to technical debt. Additionally, the three sub-questions (RQ1.1, RQ2.1, and RQ2.2) are created for helping to answer the main research questions. These dive into the difference between security debt and security vulnerabilities, how security debt is prioritized compared to other kinds of work, and the role security knowledge plays in the management of security debt.

The first research question and sub-question (RQ1 and RQ1.1) aim to get an idea of what security debt actually is and if there even is such a thing. In order to get a deeper understanding on how security debt is set apart from other security issues, RQ1.1 is created. The purpose of this sub-question is to study potential differences between security debt and security vulnerabilities. It would be interesting to find out in which cases security issues can be viewed as security debt and in which cases they are viewed as security vulnerabilities, and if there exists security vulnerabilities that are considered to be security debt.

The second research question and sub-questions (RQ2, RQ2.1, and RQ2.2) aim to find out how security debt can be managed during the software development. The two sub-questions are set to provide a deeper understanding of the management of security debt. RQ2.1 looks at how security debt is prioritized, not only compared to other types of technical debt, but also compared to feature development, bug fixes, etc. as these things are all work that needs to be done. The purpose of RQ2.2 on the other hand is to find out the role that knowledge of security plays in the management of security debt.

The goal of the third research question (RQ3) is to find the relation between security debt and technical debt. Other software qualities such as scalability have been studied as a type of technical debt (Hanssen et al., 2019). This poses the question of why not look into security debt as a technical debt?

1.3 Structure

- **Chapter 2 - Background**
  The background provides insight into security in software, the metaphor of debt, and technical debt management.
- **Chapter 3 - Methodology**
  The methodology describe the methodical choices made during the thesis which include the research design, methods of data collection, and research ethics.

- **Chapter 4 - Results**
  The results is presented according to the order of the research questions.

- **Chapter 5 - Discussion**
  In the discussion chapter, the findings from chapter 4 are discussed together with the background and related work from chapter 2. The chapter is structured according to the research questions. Finally, contributions, recommendation for practice and research are presented together with validity.

- **Chapter 6 - Conclusion**
  The main conclusion coming out of the discussion are presented in this chapter.

Chapter 2

Background

In this chapter I will describe the background for the topic of this thesis. The first section dive into security in software, the second section explain the metaphor of technical debt and its landscape, and the last section describe technical debt management.

2.1 Security in software

It is fair to assume that the identification of risk and management of software security compete for the same resources as other software development work such as feature implementation, requirements, and the fixing of bugs. The security work is often not prioritized as it is viewed as less urgent or that it does not produce visible value (Rindell et al., 2019). Many software developers still associate software quality firstly with the implementation of functionality and not security (Maymi & Harris, 2019, p. 1084). A thorough understanding of vulnerability, risk, and security is required for making informed decisions related to security in software.

When developing secure software it is central that it is of good **quality**. The reason for this is that "*quality refers to how good or bad something is for its intended purpose*" (Maymi & Harris, 2019, p. 1083). According to ISO/IEC, 2011, security is a characteristic that speaks to the quality of a software product. Having the needed security for the different systems depends on the understanding of the **environment** in which these systems exists. Understanding the environment, meaning how the environment works and what is in it is necessary for applying technologies in a controlled and comprehensive manner (Maymi & Harris, 2019, p. 1086).

McGraw, 2004 explains **software security** "*is the idea of engineering software so that it continues to function correctly under malicious attacks*" (McGraw, 2004, p. 80). It is described by Maymi and Harris, 2019, p. 1083 that developing both functionality and security together at all phases of the development life cycle will provide protection at all the needed layers of the software. The security will then be woven into the core of the software. **Secure coding** "*is the process of developing software that is free from defects, particularly those that could be exploited by an adversary to cause us harm or loss*" (Maymi & Harris, 2019, p. 1121). The adoption of **secure coding standards** can help to hold people accountable for their work. Validating input, sanitizing data, keeping the code simple, etc. are practices that can be used as part of the secure coding standard (Maymi & Harris, 2019, pp. 1122–1123). In the process of developing secure software a thorough understanding of vulnerability and risk is required.

2.1.1 Vulnerabilities and risk management

A **vulnerability** is explained as "*a weakness in a system that allows a threat source to compromise its security*" (Maymi & Harris, 2019, p. 6). **Risk** is "*the likelihood of a threat source exploiting a vulnerability and the corresponding business impact*" (Maymi & Harris, 2019, p. 7). Risk is also about uncertainty. According to Aven and Renn, 2009 *"risk refers to uncertainty about and severity of the consequences (or outcomes) of an activity with respect to something that humans value"* (Aven & Renn, 2009, p. 1). It is also explained that "*a probability is a measure of uncertainty, but uncertainties exist without specifying probabilities*" (Aven & Renn, 2009, p. 4). As a result, we need to deal with uncertainty when managing risk.

Risk management "*is the process of identifying and assessing risk, reducing it to an acceptable level, and ensuring it remains at that level*" (Maymi & Harris, 2019, p. 93). **Risk assessment** is performed as a tool for risk management and is described as "*a method of identifying vulnerabilities and threats and assessing the possible impacts to determine where to implement security* *controls*" (Maymi & Harris, 2019, p. 101). **Controls** are added in order to mitigate risks, controls can be several different countermeasures e.g. procedures where vulnerabilities are eliminated or that the likelihood of the vulnerability being exploited is reduced (Maymi & Harris, 2019, p. 7). Risk management and risk assessment is required because all environments have threats and vulnerabilities. Being able to identify these threats and assessing them is therefore important (Maymi & Harris, 2019, p. 93) for system reliability.

2.1.2 Attack surface, threat modelling, and incident management

Being aware of the possible ways the systems can be used in attacks is essential. "*An **attack surface** is what is available to be used by an attacker against the product itself*" (Maymi & Harris, 2019, p. 1093). The attack surface is modelled and analyzed for the purpose of narrowing the ways the systems can be attacked (Maymi & Harris, 2019, p. 1093). **Threat modelling** is a way to understand the valued assets and the threats against them before developing defences (Maymi & Harris, 2019, p. 97). Maymi and Harris, 2019, p. 97 define threat modelling as "*the process of describing feasible adverse effects on our assets caused by threat sources*" (Maymi & Harris, 2019, p. 97).

Attacks on software products happen. The **incident management process** consists of different phases: detection, response, mitigation, reporting, recovery, and remediation (Maymi & Harris, 2019, pp. 1000–1008). The discussion in this thesis is limited to the last phase. The remediation phase tries to make sure that that kind of attack is never successfully carried out again. The incident is reviewed in the hopes of learning how to prevent those kinds of incidents and to find out how to handle similar situations (Maymi & Harris, 2019, pp. 1008–1009).

2.1.3 Security controls

Looking into the environment in which the system will run is central for finding out how many and what kind of security controls should be implemented (Maymi & Harris, 2019, p. 1084). Implementing a security control by the means of an IT asset is called a technical control. When these technical controls are audited, the ability they have for mitigating risks is tested (Maymi & Harris, 2019, p. 871). Auditing technical controls can be done in many ways. Code reviews, penetration testing, and vulnerability testing are three of the listed approaches.

1 **Code reviews** are performed before a piece of code is pushed. During the review, it is not the author that checks the code. The reason for this is that it is not easy to find one's own typos, grammatical errors, etc. (Maymi & Harris, 2019, p. 888). Code reviews are described to be essential for making sure that the software is of good quality (Maymi & Harris, 2019, p. 1084).

2 **Penetration testing** is a process that is requested by the owner, senior management. During this process, attacks are simulated so that the environment's weaknesses can be identified and the level of resistance can be measured (Maymi & Harris, 2019, pp. 873–875).

3 **Vulnerability testing** can be automated, manual, or as a combination of the two (Maymi & Harris, 2019, p. 871). This type of assessment is for identifying vulnerabilities that are present in the environment.

The difference between penetration testing and vulnerability testing is that during a penetration test, the tester finds and exploits vulnerabilities to show the company that hackers can be able to access the systems while vulnerability testing is for identifying vulnerabilities that can be used in order to get access to the system. (Maymi & Harris, 2019, p. 878). There may exist vulnerabilities that when they are apart are not important but when they are put together they can result in a critical outcome if exploited (Maymi & Harris, 2019, p. 871).

2.1.4 Security knowledge

Barnum and McGraw, 2005, p. 74 described that "*knowledge is information in context*". Meaning that knowledge and information are in fact not the same thing, but they are related to each other. Practitioners that work with software security put a high value on both experience and knowledge. Knowledge of software security can be used and applied in many ways. Software security knowledge comes in handy in many of the phases of the life-cycle of the software, e.g. best practices that are knowledge-intensive (Barnum & McGraw, 2005). Furthermore, Assal and Chiasson, 2018 found that there is a direct connection between security knowledge and security integration; "*we found that the expectation of security knowledge (or lack thereof) directly affects the degree of security integration in developers' tasks*" (Assal & Chiasson, 2018, p. 291). In instances with limited security knowledge it was expected that the security practices were lax (Assal & Chiasson, 2018).

Sharing security knowledge "*can give a new software security practitioner access to the knowledge and expertise of all the masters*" (Barnum & McGraw, 2005, p. 74). This can be done by compiling critical security knowledge and expertise from security craftsmen (Barnum & McGraw, 2005), to make this available for the rest of the software practitioners.

The "security in software"-section provide the security background for this study. The topics in this section are relevant as they are used to describe and discuss the security perspective of the research questions. Most of the explained topics (e.g. risk management, risk assessments, attack surfaces, threat modelling, etc.) will be used to discuss how security debt should be managed. Vulnerabilities are discussed in order to find the difference between security debt and security vulnerabilities in RQ1.1, and risks and uncertainties will also be discussed during the management of security debt. Additionally, the background on security knowledge will be used when discussing RQ2.2, where the aim is to find what role security knowledge plays in the management of security debt.

2.2 Debt as a metaphor

Metaphor is understood as a "*word or phrase literally denoting one kind of object or idea is used in place of another to suggest a likeness or analogy between them*" (Merriam-Webster, n.d.). Technical debt is a metaphor used as a communication mechanism between developers and other interested parties (Avgeriou et al., 2016; Ernst et al., 2015; Kruchten et al., 2012; Sneed, 2014). Sneed, 2014 explains that the use of a monetary metaphor made it easier for business managers to understand the cost of having poor software quality. Thus, a metaphor is a rhetoric instrument to increase focus on sub-optimal solutions. Another rhetoric instrument is securitization. When introducing security debt into the technical debt discussions, security aspects will gain importance, a process also called securitization. Securitization is understood as applying a rhetoric of threat to issues in order to increase security priorities (Buzan et al., 1998).

2.2.1 Technical debt

Avgeriou et al., 2016 defined technical debt as "*a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability*" (Avgeriou et al., 2016, p. 112).

The three main concepts that are included in this definition are: debt, interest, and principal (Avgeriou et al., 2016):

- The **debt** is the existence of none-optimal solutions in a system.
- A debt has an additional cost, an **interest**, that have to be repaid because of the debt.
- The **principal** is described as the cost in order to either refactor the system to remove the accumulated debt, or the cost of developing the system without the debt.

Fowler, 2009 looked further into the technical debt metaphor and pointed out that it could be divided into four sections; reckless/prudent and deliberate/inadvertent, giving a quadrant. Figure 2.1 shows the quadrant. Below are explanations for each section of the quadrant (Fowler, 2009):

- **Reckless-deliberate**: reckless debt is due to messiness, which means that the mess results in a lot of interest that needs to be repaid. Reckless-deliberate debt occurs when a development team is aware of good design practices but chooses not to follow them because they do not think they have the time for writing clean code, although clean code and a good design helps with faster development.
- **Reckless-inadvertent**: reckless debt can also be inadvertent, meaning that a team is ignorant to good practices. The accumulation of technical debt is then due to them not realizing that they are taking on the debt.

- **Prudent-inadvertent**: Prudent debt is not due to mess but rather choices that are made in regards to design flaws. Prudent-inadvertent debt occurs when an experienced development team follow a good design strategy but finds out during the development what kind of design strategy they should have chosen. This realization is prudent-inadvertent debt and even the best development teams will accumulate technical debt when working on a project.
- **Prudent-deliberate**: If the interest produced by the choices that are made are small enough then it might not be something worth repaying, meaning that it is about whether or not the short-term benefit is worth the repayment needed in the future.

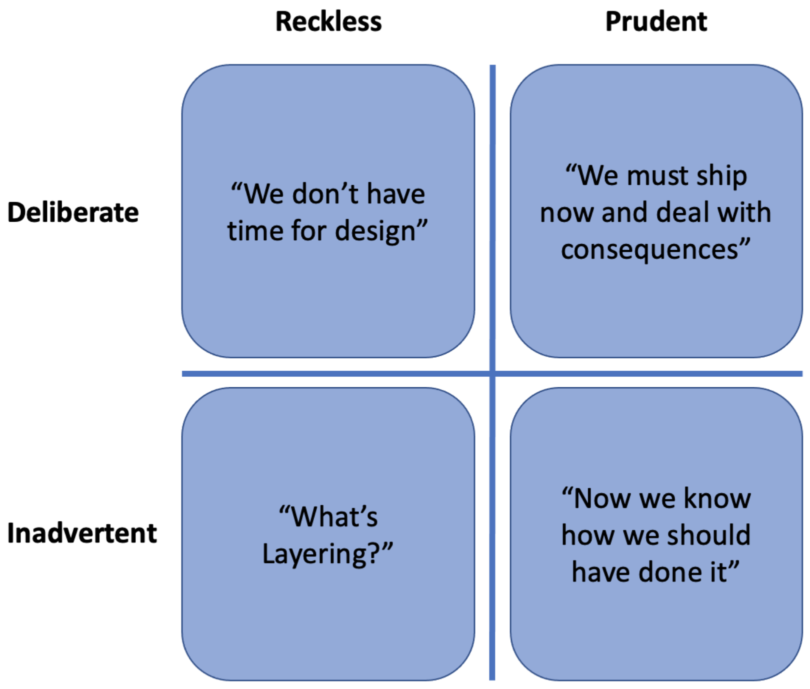

Reckless
Prudent
Deliberate
"We don't have time for design"
"We must ship now and deal with consequences"
Inadvertent
"What's Layering?"
"Now we know how we should have done it"

Figure 2.1: Flower's quadrant presented in Fowler, 2009

In addition to Fowler explaining technical debt as a quadrant based on factors such as awareness (reckless/prudent) and intention (deliberate/ inadvertent), point Kruchten et al., 2012 out that technical debt also can be due to change in technology. What is meant by this is that gaps in technology is not generated internally as a result of a wrong choice but rather externally as a result of the change in context (Kruchten et al., 2012). It is something that can be seen in retrospect.

Technical debt causes poor product quality, increased cost and can slow the progress of the software development which can affect its success in the long term (Lim et al., 2012). Quality attributes such as maintainability, performance and usability, reliability, security, etc. are impacted by technical debt (Li et al., 2015; Tom et al., 2013).

There are many kinds of technical debt that make out the technical debt landscape. Li et al., 2015 found in their systematic mapping study ten types of technical debt: requirement TD, architectural TD, design TD, code TD, test TD, build TD, documentation TD, infrastructure TD, version TD, and defect TD. Alves et al., 2016 found an additional five types of technical debt: people debt, process debt, automation test debt, usability debt, and service debt. In their tertiary study, Rios et al., 2018 found the same 15 technical debt kinds by studying a number of systematic literature reviews. Alves et al., 2016 and Rios et al., 2018 found that the most researched types of technical debt was design debt, architecture debt, documentation debt, test debt, and code debt. An additional type of technical debt will be presented in subsection 2.2.1. Three of these types of technical debt are described below.

Architectural technical debt

Architectural technical debt is the kind of technical debt that is "*incurred at the architectural level of software design*" (Verdecchia et al., 2020, p. 1) and "*is caused by architecture decisions that make compromises in some internal quality aspects, such as maintainability*" (Li et al., 2015, p. 201). It is found that architectural technical debt is the most difficult technical debt type to uncover (Kruchten et al., 2012).

Verdecchia et al., 2021 described the relation between security and architectural technical debt as the following:
"*security breaches are a recurrent symptom of ATD. Due to the complexity caused by the ATD present in a software-intensive system, inadvertent security flaws can be introduced, leading to the unintentional disclosure of private information to unauthorized parties. Such data leaks can be a strong signal of ATD, that has to be tackled with a reactive management strategy ... as soon as the symptom arises*" (Verdecchia et al., 2021, p. 14).

Defect debt

Akbarinasaji et al., 2016 described defect debt as "*the trade-off between the short-term benefit of postponing bug fixing activities and long-term consequence of delaying those activities*" (Akbarinasaji et al., 2016, p. 1). This means that all identified defects, failures, and bugs that are not fixed during the same release as they are found is considered defect debt. It is important to both measure and monitor defect debt as it can cause system bankruptcy due its accumulation in the issue tracking system (Akbarinasaji et al., 2016). "*The concept of bankruptcy as it is applied to technical debt refers to a situation of overwhelming debt interest, whereby progress is halted and a complete rewrite is deemed necessary*" (Tom et al., 2013, p. 1503).

Social debt and lack of knowledge

In addition to the 15 kinds of technical debt found by Rios et al., 2018, have something called social debt been studied. Tamburri et al., 2013 used four scenarios in order to exemplify social debt. The four scenarios followed the visualization of technical debt quadrant (Kruchten et al., 2012) where the fourth section shows social debt as invisible and that is has a negative effect. This fourth scenario describes a situation where a legacy product is to be renovated where the original development knowledge and expertise is no longer available/is lost. "*In this case, the socio-technical decision to renovate without the original community and its knowledge exposes an invisible and negative effect: a social debt*" (Tamburri et al., 2013, p. 94). The community that is currently developing on the legacy product is sub-optimal as they do not have the needed knowledge and therefore have to equalise the debt by for example reverse-engineering the lost knowledge (Tamburri et al., 2013).

Martini et al., 2015 listed factors/causes for the accumulation of architectural technical debt where one of the causes is lack of knowledge. The lack of knowledge was divided into inexperience, lack of domain knowledge, ignorance, and carelessness. All these sub-factors can contribute to sub-optimal decisions resulting in architectural technical debt (Martini et al., 2015, p. 246):

- **Inexperience**: "*New employees are more subjected to accumulating ATD due to the natural non-complete understanding of the architecture and* *patterns*".

- **Lack of domain knowledge**: "*This factor might be related to the previous one, or, as the informants mentioned, to the generalization of Agile teams, which might need to develop a feature accessing a complex component of which they do not have expertise*".
- **Ignorance**: "*Lack of knowledge about where the architectural rules are stored (documentation)*".
- **Carelessness**: "*Lack of awareness of the importance of architecture. A recurrent statement from the informant is that having documentation is not enough to avoid architecture violations*".

Thus, knowledge is an important part of software development and for the system's security. Siavvas et al., 2019, p. 1 found evidence that technical debt can "*be used as an indicator of software security*".

2.2.2 Security debt

Security debt is a term that has been attracting increasing attention by practitioners and researches as the technical debt management is getting more mature (Martinez et al., 2021). Security debt have been described in research throughout the last years but a formal definition have yet to be created.

In 2016, security debt was described as "*a particular solution that compromises the security of the system, e.g., introducing a particular breach of security*" (Silva et al., 2016). Then in 2019, Rindell et al., 2019 explained security debt as "*a technical debt containing a security risk*" (Rindell et al., 2019) and Rindell and Holvitie, 2019 explained a two-fold defining of security debt: (1) technical debt found through security verification or validation methods, and (2) technical debt in a software component that is critical to the security. Martinez et al., 2021, p. 2 created the latest definition on the basis of 22 publications, including the three explanations above: "*security debt is incurred when limited approaches or solutions are applied (intentionally or unintentionally) to reach the needed security levels for the system in operation*". Having security debt in a system makes it more vulnerable to attacks (Martinez et al., 2021).

Both Rindell et al., 2019 and Siavvas et al., 2019 discuss the relationship between technical debt and security. Rindell et al., 2019 described that security debt has a relation to technical debt and security risks. Looking at security risks and technical debt together would make it possible to use security engineering techniques for finding them as these techniques provide a way of finding issues in the internal quality related to requirements, architecture, coding, and testing.

This "debt as a metaphor"-section has described the information needed to discuss the debt perspective of the different research questions. When discussing how security debt is defined, the information gathered on earlier research on security debt will be used. Additionally, the definition of technical debt and the quadrant by Fowler, 2009 will be used to discuss the relation between security debt and technical debt (RQ3). Defect debt will be used in order to discuss the difference between security debt and security vulnerabilities. Social debt and the lack of knowledge, together with security knowledge, will be used to discuss the role security knowledge plays on the different security debt management activities. RQ3 aim to find the relation between security debt and technical debt as described above. In this discussion, architectural technical debt will also be used to discuss the relation between security debt and architectural technical debt.

2.3 Technical debt process

Technical debt can be an investment as long as the team working on the system is aware of (have the required knowledge) the technical debt that is being accumulated. Additionally, the increased risk to the system due to the technical debt should also be known of and tracked (Kruchten et al., 2012). Technical debt is found to be detrimental (Lim et al., 2012), so if not managed properly, technical debt can cause a lot of problems e.g. financial problems and technical problems. Such problems can lead to maintenance and evolution costs increasing (Nord et al., 2012). For keeping track of the technical debt it needs to be managed properly.

Technical debt management (TDM) consists of several activities for preventing the occurrence of potential technical debt or for handling the existing technical debt so that it is kept at a reasonable level (Li et al., 2015). Ampatzoglou et al., 2015; Li et al., 2015; Rios et al., 2018 are three studies that found a number of TDM activities. The activities found by each publications can be seen in table 2.1. The additional three activities visualisation, time-to-marked analysis, and scenario analysis found by Rios et al., 2018 were described by Fernández-Sánchez et al., 2017 as elements. In that paper they differentiated between activities and elements: "*the elements, as defined within this paper, conceptually different from activities, are used during the execution of activities as inputs, outputs, or mechanisms*" (Fernández-Sánchez et al., 2017, p. 23).

Activity	**Ampatzoglou et al., 2015**	**Li et al., 2015**	**Rios et al., 2018**
Prevention	-	x	x
Identification	x	x	x
Documentation/ representation	-	x	x
Visualization	-	-	x
Communication	-	x	x
Measurement	x	x	x
Prioritization	x	x	x
Monitoring	x	x	x
Repayment	x	x	x
Time-to-market analysis	-	-	x
Scenario analysis	-	-	x

Table 2.1: Technical debt management activities found in the three earlier publications Ampatzoglou et al., 2015; Li et al., 2015; Rios et al., 2018

Strategies (Rios et al., 2018), approaches (Li et al., 2015), actions (Freire et al., 2020), and practices (Pérez et al., 2021) have been described to be utilized during TDM activities. Even though different publications use different words, they have some overlapping aspects, e.g. cost-benefit analysis have been found by Alves et al., 2016; Ampatzoglou et al., 2015; Li et al., 2015 and described as both approaches and strategies. There are a lot of approaches and strategies that are not relevant for this study and will therefore not be included. The following eight sub-sections corresponds to the eight activities found by Li et al., 2015 presented in table 2.1. After these activities, follows a discussion on specific approaches/tools that can be used for different types of technical debt, including security related approaches for security debt (see sub-section 2.3.9).

2.3.1 Prevention

Technical debt prevention aims to "*prevent potential TD from being incurred*" (Li et al., 2015, p. 204). Li et al., 2015 mentioned four approaches for preventing technical debt, Pérez et al., 2021 found nine practices for preventing technical debt, and Freire et al., 2020 highlighted the 5 most cited preventative actions in their study. The most relevant (for this study) ways of preventing the accumulation of technical debt mentioned in these publications are:

- **Human factors analysis**: "*cultivate a culture that minimizes the unintentional TD caused by human factors, e.g., indifference and ignorance*" (Li et al., 2015, p. 208).
- **Adoption of good practices** is discussed by both Freire et al., 2020; Pérez et al., 2021. Pérez et al., 2021, p. 8 described the adoption of good practices as "*related to code development and covers the following subpractices: following a well-defined development standards and adoption of pair programming*".
- **Code evaluations/standardization**: "*this practice can go hand in hand with the use of tools to perform a continuous inspection of the code quality to detect bugs, code smells, and security vulnerabilities*" (Pérez et al., 2021, p. 8).
- **Training (code review/refactoring)**: "*training on code reviews could be useful to reduce TD occurrence. On the other hand, training on refactoring is more related to TD payment practices*" (Pérez et al., 2021, p. 8).

2.3.2 Identification

Technical debt "*identification detects TD caused by intentional or unintentional technical decisions in a software system through specific techniques, such as static code analysis*" (Li et al., 2015, p. 204). Two of the four approaches mentioned by Li et al., 2015, p. 206 for identifying technical debt are:

- **Code analysis**: "*analyze source code to identify violations of coding rules, lack of tests; calculate software metrics based on source code to identify design or architecture issues*".
- **Check list**: "*check against a list of predefined scenarios where TD is*

incurred".

Tools for identifying technical debt have also been found, e.g. DebtFlag, FindBugs, Sonar TD plugin, CodeVizard, SonarQube (Li et al., 2015). Tools such as Coverity and SonarQube have been mentioned to be used when prioritizing technical debt (Lenarduzzi et al., 2021). Coverity and SonarQube will be further discussed later in this study.

2.3.3 Representation/Documentation

Technical debt "*representation / documentation provides a way to represent and codify TD in a uniform manner addressing the concerns of particular stakeholders*" (Li et al., 2015, p. 205). Saraiva et al., 2021 mentioned three project management tools for documenting technical debt: Hansoft, Jira, Redmine. When documenting the technical debt, a number of fields (15 fields) for describing the technical debt and its context are proposed (Li et al., 2015, p. 207). Some of them are presented below because of their relevance to the study:

- **ID**: "*a unique identifier for a TD item*".
- **Location**: "*the location of the identified TD item*".
- **Responsible/author**: "*the person who is responsible for repaying the TD item*".
- **Type**: "*the TD type that this TD item is classified into, e.g., architectural TD*".
- **Description**: "*general information on the TD item*".

Technical debt is not only something that can be marked with "TODO" or "fixme" in the code but can also be added to issue tracking systems. In these systems the technical debt (sub-optimal implementation decision) can be documented with labels such as "technical debt" or "debt" (Xavier et al., 2020).

2.3.4 Measurement

Technical debt "*measurement quantifies the benefit and cost of known TD in a software system through estimation techniques, or estimates the level of the overall TD in a system*" (Li et al., 2015, p. 204). 17 strategies for measuring technical debt was found by Rios et al., 2018 and 6 approaches was found by Li et al., 2015. Two relevant measurement approaches for this study are:

- **Cost-benefit analysis**: whereas Rios et al., 2018 just mention the cost-benefit analysis, Alves et al., 2016 describes it in more detail as an analysis to find out if the interest that is expected is large enough to justify repaying the debt. There are two parts to the rate of the interest: 1) the probability that the non-paid debt will result in extra costs, and 2) a calculated amount of extra work that needs to be done if the debt is not repaid (Alves et al., 2016).
- **Human estimation**: "*estimate TD according to experience and expertise*" (Li et al., 2015, p. 207).

2.3.5 Monitoring

Technical debt "*monitoring watches the changes of the cost and benefit of unresolved TD over time*" (Li et al., 2015, p. 204). Three of the approaches mentioned for monitoring technical debt are (Li et al., 2015, p. 208):

- **Planned checks**:"*regularly measure identified TD and track the change of the TD*".
- **TD monitoring with quality attribute focus**: "*monitor the change of quality attributes that detrimental to TD, such as stability*".
- **Threshold-based approach**: "*define thresholds for TD related quality metrics, and issue warnings if the thresholds are not met*".

A tool used for quantifying code metrics as a means for monitoring is SonarQube (Saraiva et al., 2021).

2.3.6 Communication

Technical debt "*communication makes identified TD visible to stakeholders so that it can be discussed and further managed*" (Li et al., 2015, p. 205). Communication is one of the activities that are continuously carried out during the TDM (Rios et al., 2018). Li et al., 2015, p. 208 identified six approaches for communicating technical debt. Three relevant approaches to communication are:

- **TD dashboard**: "*A dashboard displays TD items, types, and amounts in order to get all stakeholders informed of the existence of the TD*".
- **Backlog**: "*All identified TD items as well as anything to be resolved in the development are put into the backlog of the software project, so that the TD items can be treated as important as known bugs and unimplemented planned features and functionalities*".
- **TD list**: "*A TD list keeps all identified TD items and make them visible to stakeholders*".

2.3.7 Prioritization

Technical debt "*prioritization ranks identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases*" (Li et al., 2015, p. 204). Due to limited resources allocated to the repayment of technical debt is it not possible to repay all the system's technical debt (Fernández-Sánchez et al., 2017). Rios et al., 2018 found 13 strategies for prioritizing technical debt, Li et al., 2015 found 4 approaches to technical debt prioritization, and Lenarduzzi et al., 2021 explained 5 aspects of the prioritization and a number of strategies. Two of the relevant approaches are:

- **Analytic hierarchy process (APH)**: Rios et al., 2018 found APH as a prioritization approach while Alves et al., 2016 described it as as way to rank the different technical debts, where the ranking gives an indication to what technical debt should be repaid first. This strategy is based on different identified technical debt instances (Alves et al., 2016).
- **Portfolio approach**: Rios et al., 2018 found the approach but it is described by Li et al., 2015 and Guo and Seaman, 2011. "*The portfolio approach considers TD items along with other new functionalities and bugs as risks and investment opportunities (i.e., assets)*" (Li et al., 2015, p. 206). "*The goal of portfolio management is to select the asset set that can maximize the return on investment or minimize the investment risk*" (Guo & Seaman, 2011, p. 32).

Both the portfolio approach and cost-benefit analysis have been found to be used during the TDM activities measurement, monitoring, and prioritization (Rios et al., 2018).

Lenarduzzi et al., 2021, p. 7 performed a literature review where they identified strategies, processes, factors, and tools for the prioritization of technical debt and mentioned five "*themes illustrating different prioritization aspects*":

1) **Internal software quality**: the publications that had a focus on internal software quality also focused on software quality assessments for finding technical debt that caused the most amount of maintenance cost. Additionally, factors such as the remaining product life, severity of the debt and the impact it has on the future development activities, and the constraints that are related to business was taken into consideration.
2) **Software productivity**: considering the decrease in productivity is also something that is done when prioritizing technical debt.
3) **Software correctness**: prioritizing the technical debt that has the greatest negative effect on software correctness.
4) **Cost-benefit analysis**: finding the most lucrative prioritization of the technical debt and the possible refactoring activities.
5) **Combinations of approaches**: combining approaches can also be done, e.g. combining Analytic Hierarchy Process (AHP), the Portfolio method, etc..

In the study Lenarduzzi et al., 2021, 12 tools for prioritizing technical debt were named. The static analysis tools most central for this thesis are Coverity and SonarQube. If the code is not in compliance with a set of rules, technical debt issues are raised. When this happens, the issue's severity is provided together with a classification, for example that it can lead to bugs or to security vulnerabilities (Lenarduzzi et al., 2021).

Ribeiro et al., 2017 found a multiple decision strategy criteria model where several prioritization approaches are used. This model can be used in many phases in the project and has four categories of decision criteria (Ribeiro et al., 2017, pp. 335–336):

1) **Nature**: "*criteria that are related to TD's properties, such as their severity and time when the debt was incurred*".
2) **Customer**: "*criteria into this category concern about the impact that debt items have on customers*.
3) **Effort**: "*criteria that are related to the cost of TD, such as the impact of TD on the project and what effort will be necessary to pay the debt item off, and*".

4) **Project**: "*criteria that are related to projects' properties, such as their lifetime and their possibility of evolution*".

Tools can also help with the prioritization of technical debt. AnaConDebt, Coverity, and SonarQube are thee of them (Lenarduzzi et al., 2021). AnaConDebt is a tool that provides a value that is easy to understand and is comparable to other values. Having these comparable values can help with the prioritization (Martini & Bosch, 2017). The static analysis tools Coverity and SonarQube can also help with the prioritization (Lenarduzzi et al., 2021). Static analysis tools "*analyze the code without actual code execution to uncover defects and provide a characterization of the code, e.g., size and quality attributes*" (Eisenberg, 2012, p. 1).

2.3.8 Repayment

Technical debt "*repayment resolves or mitigates TD in a software system by techniques such as reengineering and refactoring*" (Li et al., 2015, p. 205). Li et al., 2015 identified seven approaches for technical debt repayment and Rios et al., 2018 identified two strategies for the repayment.

- **Refactoring**: "*Make changes to the code, design, or architecture of a software system without altering the external behaviors of the software system, in order to improve the internal quality*" (Li et al., 2015, p. 208).
- **Reengineering**: "*Evolve existing software to exhibit new behaviors, features, and operational quality*" (Li et al., 2015, p. 208).
- **Repackaging**: "*Group cohesive modules with manageable dependencies to simplify the code*" (Li et al., 2015, p. 208).
- **Rewriting**: "*Rewrite the code that contains TD*" (Li et al., 2015, p. 208).
- **Bug fixing**: "*Resolve known bugs*" (Li et al., 2015, p. 208).
- **Partial refactoring for ATD**: Rios et al., 2018 found the approach but it is explained by Martini et al., 2015. In this approach, the technical debt is partially refactored, as complete refactoring is not possible and little to no refactoring can result in a development crisis. Partial refactoring then leads to the most amount of refactoring (Martini et al., 2015).

2.3.9 Differences in the technical debt process

The 8 activities in the technical debt process have been described above with a number of approaches. There have been found specific approaches and tools directed to specific types of technical debt. Some of the differences relevant for this study are tools, approaches, and security.

Tools
Saraiva et al., 2021 found in their systematic mapping that 34% of the tools could be used for several types of technical debt (e.g. SonarQube), meaning that they are not dedicated to one type of technical debt in particular. 60% of the tools they found, on the other hand, was found to be tailored to specific types of technical debt. Saraiva et al., 2021 also found an increase in the use of tools when it comes to architectural technical debt. They found that 40% of the tools in their systematic mapping could support architectural technical debt.

Approaches
Alfayez et al., 2020 performed a systematic literature review on **prioritization approaches** and found that $\approx 71\%$ of the identified technical debt prioritization approaches works for specific types of technical debt while the rest, $\approx 29\%$, can work for any type of technical debt. The highest number of prioritization approaches was found to address the prioritization of code debt, design debt, and defect debt.

Freire et al., 2020 used data from the InsighTD [1] [2] project and performed analysis for finding the most used preventative actions in practice. Their findings showed that several **preventative actions** such as "following the project planning" and "training" was used for several types of technical debt. It was also found specific preventative actions for specific types of technical debt, e.g. "using the most appropriate technology version" is something that can be done for preventing architectural technical debt, "code standardization" is for preventing code debt, and documentation debt has the preventative action called "well-defined documentation".

Both security (ISO/IEC, 2011) and technical debt (Kruchten et al., 2012) are

[1]"*A family of industrial surveys specifically designed to study software engineering TD*" (Freire et al., 2020).

[2]http://www.td-survey.com

connected to a system's internal quality. As this is the case can approaches for TDM also be used for security risk management, e.g. the portfolio approach (Rindell et al., 2019). Rindell and Holvitie, 2019 explains that their way of defining security debt is in direct relation to technical debt. Using the portfolio approach for security debt requires extra input; risk (as a product of probability and impact) "*or a similar quantitative risk assessment measure*" (Rindell & Holvitie, 2019). Siavvas et al., 2019 found in their empirical evaluation "*evidence for the ability of TD to indicate security risks in software products*" (Siavvas et al., 2019) as described above.

Rindell et al., 2019 explained that "*security engineering techniques provide an effective method to recognize internal quality issues in software requirements, architecture, coding, and testing*" (Rindell et al., 2019) and that "*software security engineering offers a wide set of techniques and tools directly applicable for the management of technical debt*" (Rindell et al., 2019). Rindell and Holvitie, 2019 found that "*managing the security risk as debt provides new means for security risk mitigation in software development. Security debt can be directly integrated into existing technical debt management frameworks and tools with few technical adjustments"*. A significant addition here would be a security risk management process (Rindell & Holvitie, 2019).

The described activities, approaches, and tools that are used to manage technical debt will be used when discussing the security debt process. This section contributes with the technical point of view on how security debt should be managed. The activities and approaches presented in this section will be discussed together with security related topics, e.g. risk management, secure coding standards, threat modelling, etc.. Together they will help discuss the process for handling security debt.

Chapter 3

Methodology

In this chapter I will present the methodical choices made during the project. First I will present the research design, then I will discuss the research approach and methods for data collection. I will end the chapter with describing how the data was analyzed and the ethics surrounding the research.

3.1 Company context

The company used in this study is a conglomerate of multiple multinational companies that are organized using self managed teams. They have global processes that can be used by each company, still each team can adapt these processes to their needs. To have a unified way of measuring the self managed teams, the company uses a maturity index system which is based on penalty points. This can be seen as a way to help the teams decide and prioritize some of the work they do, like security.

3.2 Research design

The main focus of this study is finding out how service/system architects view security debt, how they think it should be managed, and how security debt relates to technical debt. It is then essential to have a research design that allows for getting the needed data from the practitioners. A visual representation of the research design is presented in figure 3.1. There are four different elements in the figure and they are described in "Legend". The three boxes with different shades of blue represents data collection, methods, and analysis and the white oval element represents the results.

As shown in figure 3.1, there are three phases of the data collection. In **phase 1**, a preliminary study was performed. The method used was literature review for gathering current literature on security debt. The result of the literature review showed a lack of a formal security debt definition, limited research on security debt management, and a possible relation between security debt and technical debt. This was the groundwork for the detailed formulation of the research problem (RP), the research questions (RQs), and the interview guide. The results from the literature review brought me to **phase 2**, qualitative data collection. This data collection was divided in two: document study and semi-structured interviews. I received a document containing the company's technical debt process, and this document was kept in mind when conducting the interviews and was used during the thematic analysis. The second method used during phase 2 was semi-structured interviews. Following the completion of the interviews and the transcriptions, two things happened: 1) the transcribed information was ready for analysis and 2) gaps in the collected information were found. These gaps were filled by **phase 3** of the data collection. This third data collection gathered the last bits of information by using email before all information from the respondents were analysed. The thematic analysis produced 7 themes based on the coding of the respondents answers. From the presented themes the goal is to provide an understanding of security debt, its management, and its relation to technical debt. In order to answer this, the results from the literature review is discussed together with the results from the thematic analysis. This is visualized by arrows pointing from the first oval element (after the literature review) and the 7 oval elements (after the thematic analysis), both pointing to the last oval element. More detailed explanations of the phases will be presented later in this chapter.

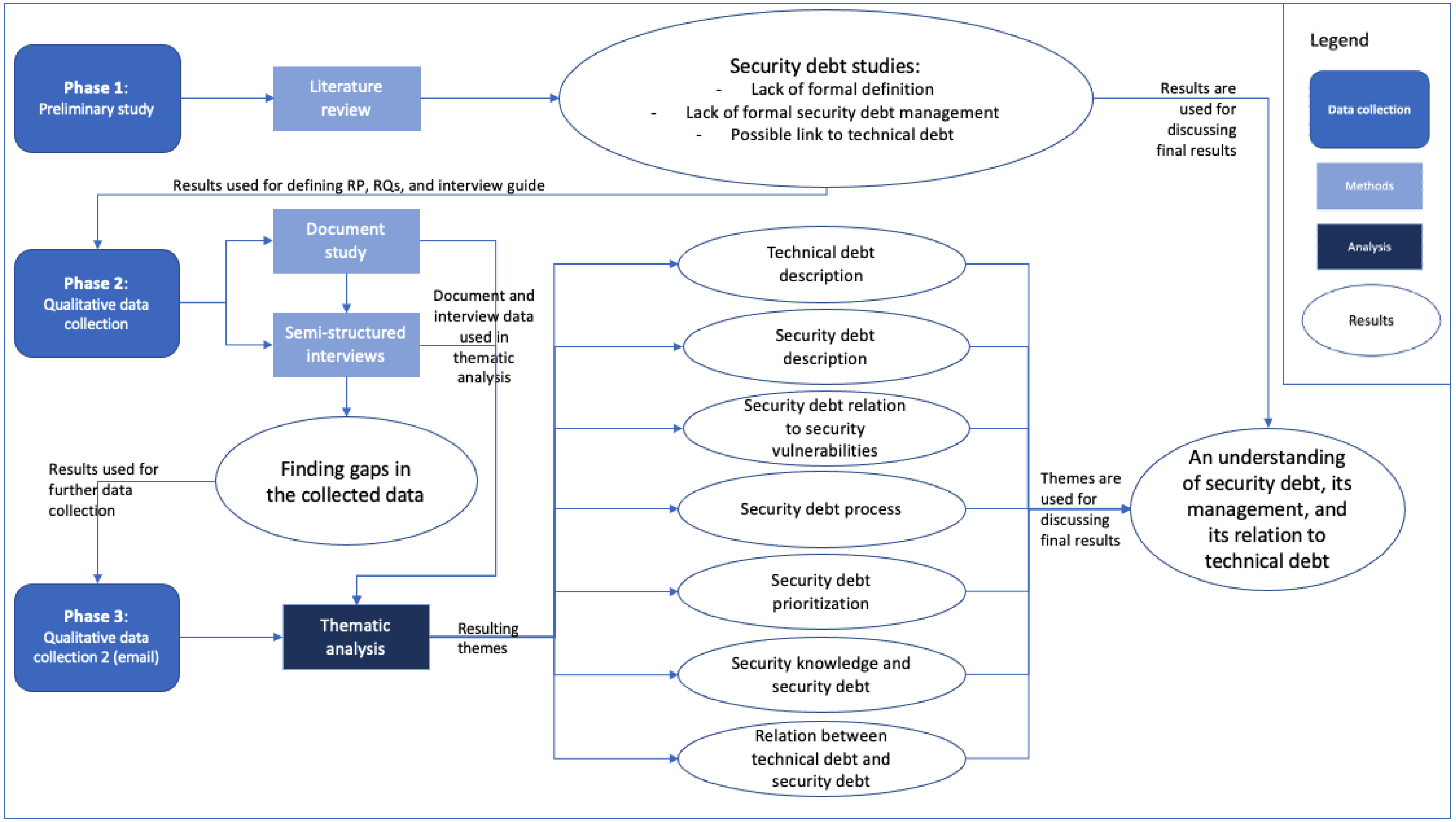

Phase 1:
Preliminary study
Literature review
Security debt studies:
- Lack of formal definition
- Lack of formal security debt management
- Possible link to technical debt
Results are used for discussing final results
Results used for defining RP, RQs, and interview guide
Phase 2:
Qualitative data collection
Document study
Semi-structured interviews
Document and interview data used in thematic analysis
Finding gaps in the collected data
Results used for further data collection
Phase 3:
Qualitative data collection 2 (email)
Thematic analysis
Resulting themes
Technical debt description
Security debt description
Security debt relation to security vulnerabilities
Security debt process
Security debt prioritization
Security knowledge and security debt
Relation between technical debt and security debt
Themes are used for discussing final results
An understanding of security debt, its management, and its relation to technical debt
Legend
Data collection
Methods
Analysis
Results

Figure 3.1: Research design model displaying the three phases of data collection, the methods of data collection, and data analysis. Descriptions of the elements used in the model are explained in "Legend".

3.3 Qualitative research design and data collection method

There are different kinds of qualitative research designs, two of which are grounded theory and case study (Creswell & Creswell, 2018, pp. 13–14). In addition to these two, action research is another qualitative research design (Wohlin et al., 2012, p. 56).

- A **grounded theory** is "*a design of inquiry from sociology in which the researcher derives a general, abstract theory of a process, action, or interaction grounded in the views of participants*" (Creswell & Creswell, 2018, p. 13).
- In software engineering a **case study** "*is an empirical enquiry that draws on multiple sources of evidence to investigate one instance (or a small number of instances) of a contemporary software engineering phenomenon within its real-life context, especially when the boundary between phenomenon and context cannot be clearly specified*" (Runeson et al., 2012).
- The purpose of **action research** is to "*influence or change some aspect of whatever is the focus of the research*" (Robson, 2002). A more detailed explanation is presented by Greenwood and Levin, 2006 as "*a set of self-consciously collaborative and democratic strategies for generating knowledge and designing action in which trained experts in social and other forms of research and local stakeholders work together.*"

Thus, a difference between case study and action research is that action research has both a focus on and is involved in the process of change. Case studies on the other hand is more observational (Wohlin et al., 2012, p. 56). Grounded theory does not only look at processes, but also actions, and interactions (Creswell & Creswell, 2018, p. 13) while I am studying a software engineering phenomenon. Based on the research problem described in section 1.2 I have come to the conclusion that the qualitative research methodology **case study** is the best way to provide answers. The reason for this is that a case study is fitting when studying a contemporary phenomenon such as security debt as it gives a more in depth understanding of the studied phenomenon (Runeson & Höst, 2009, p. 132), within the context of a software company. The thesis is also of an **exploratory** nature (Grenness, 1997) due to the limited research on security debt and management.

I conducted a number of interviews with software/system architects from one software company to collect data on how they understand and manage security debt. Interview as a data collection method is important in case studies (Runeson & Höst, 2009, p. 145). Three ways of structuring interviews are structured interviews, semi-structured interviews, and unstructured interviews (Robson, 2002). Runeson and Höst, 2009, p. 145 describe the three forms of interviews:

- When performing **structured interviews**, "*all questions are planned in advance and all questions are asked in the same order as in the plan*".
- During **semi-structured interviews**, the "*questions are planned, but they are not necessarily asked in the same order as they are listed*".
- When doing **unstructured interviews**, "*the interview questions are formulated as general concerns and interests from the researcher*".

I performed **semi-structured interviews** as this kind of interview structure is common when doing case studies. Semi-structured interviews makes it possible to do some improvisation and exploration during the interviews (Runeson & Höst, 2009, pp. 145–146).

3.4 Research approach

The three **approaches to research** are quantitative research, qualitative research, and mixed methods research. When choosing the research approach, things such as the research problem, the researchers' philosophical assumptions, the research designs, and what research methods to use, etc. should be taken into consideration (Creswell & Creswell, 2018, p. 3).

The simplest way of describing the difference between quantitative and qualitative research is that quantitative research produces numbers while qualitative research produces words. Mixed methods on the other hand is a combination of the two, where elements from both quantitative and qualitative approaches are used (Creswell & Creswell, 2018, p. 3);

- The **qualitative research** approach "*is an approach for exploring and understanding the meaning individuals or groups ascribe to a social or* *human problem. The process of research involves emerging questions and procedures, data typically collected in the participant's setting, data analysis inductively building from particulars to general themes, and the researcher making interpretations of the meaning of the data.*" (Creswell & Creswell, 2018, p. 4).

- The **quantitative research** approach "*is an approach for testing objective theories by examining the relationship among variables. These variables, in turn, can be measured, typically on instruments, so that the numbered data can be analyzed using statistical procedures.*" (Creswell & Creswell, 2018, p. 4).
- The **mixed methods research** approach "*is an approach to inquiry involving collecting both quantitative and qualitative data, integrating the two forms of data, and using distinct designs that may involve philosophical assumptions and theoretical frameworks*" (Creswell & Creswell, 2018, p. 4).

The approach was based on what kind of data I was looking to collect: I wanted to produce words and not numbers. The aim of the study is to find out what the security debt phenomenon is and how it is managed, and thus it is important to talk to software practitioners to produce general themes and interpret the collected data. Based on the research problem and the research questions, the exploratory case study design, and semi-structured interviews, pointed in the direction of a qualitative approach.

3.5 Methods for data collection

A literature review and interviews form the basis for data collection in this study.

3.5.1 Literature review

I conducted a literature review to provide an understanding of the current literature on security debt and technical debt. A literature review "*provides a framework for establishing the importance of the study as well as a benchmark for comparing the results with other findings*" (Creswell & Creswell, 2018,

p. 26). The review laid the foundation for the formulation of the research problem and the research questions, and later for the creation of the interview guide.

In order to find papers on security debt, I used the search engine Google Scholar [1]. Table 3.1 shows the initial search words used for finding articles that related to security debt. As shown in the table, there is a decline in number of hits when the search becomes increasingly specific. From what I have seen, there is no consistent use of only one term for the phenomenon studied in this thesis. Therefore an "OR" statement between "security debt" and "security technical debt" was added in order to find all studies that mention either of them. The last row in the table has the least amount of hits. Here I specifically searched for "security debt" and "technical debt" in a software context. This search limits the amount of hits as studies that strictly mention "security technical debt" are not found. Additionally, when discussing security debt, it is often in the context of technical debt so this is a term that most likely is used when discussing security debt. It is also important to remember that included in the number of hits are publications that only mention "security debt" or "security technical debt" in the title and in the abstract but not necessarily in the rest of the study. The majority of the papers found came from IEEE, ResearchGate, Wiley, acm, and ScienceDirect.

Search words	**Number of hits**	**Include citations**
security debt	2 970 000	Yes
"security debt"	1 800	Yes
"security debt"	1 730	No
"security debt" OR "security technical debt"	1 750	No
("security debt" OR "security technical debt") +software	275	No
("security debt" OR "security technical debt") +software -"social security debt"	194	No
("security debt" OR "security technical debt") +"technical debt" +software	51	No
"security debt" +"technical debt" +software	35	No

Table 3.1: Hits per search on Google Scholar.

[1]https://scholar.google.no

3.5.2 Interviews

Before the interviews, I needed some preliminary information regarding the technical debt process so that I could easier keep up with the respondents during the interviews. A meeting with a company employee was set up where they explained the process and after the meeting I was sent a document of the process. This initial preparation assisted me in the creation of the interview guide. The interview guide is presented in the appendix.

I chose to perform semi-structured interviews, as explained earlier. After three interviews I found that the initial interview guide had to be more specific. The same type of information was gathered but the interview guide had to be expanded in order to get in-depth information. In addition to data on the degree to which they identified technical debt, I also needed information on how they identified technical debt (by hand or with the use of tools). As I performed semi-structured interviews, this was something I was able to do as these types of interviews open up for improvisation and exploration (Runeson & Höst, 2009, pp. 145–146).

After wrapping up the interviews I transcribed them and found that I got some important information during the last interviews that was not discussed during the initial interviews (e.g. "Are there any cases/scenarios where the fixing of security debt is not prioritized? Please explain a little bit."). Therefore, I sent out customized emails with follow-up questions (Phase 3). This also included getting the needed information from the first three respondents before the change to the interview guide. 21 of my respondents replied with answers to the follow-up questions.

I set aside one hour for each interview session. Some of the interviews lasted 45 minutes while others lasted for more than one hour. This depended on how fast the respondents answered and what they said. My interviews were structured according to the four phases presented by Runeson and Höst, 2009, p. 146:

1) I explained that the interview would be recorded, I fixed consent, and had a brief description of what we would be talking about,
2) Introductory questions to establish background,
3) The central part of the interview where I got the main information for

answering the research questions, and

4) I rounded up the interview, described that I would send the respondent(s) some additional information about the meeting/thesis, thanked them for participating, and spoke of follow-up emails.

Both during the creation of the interview guide and during the interviews I had a goal for each question in the interview guide in mind. This made it easier to follow the discussion during the interview.

The interviews were conducted on Zoom as there is an ongoing pandemic and the majority of the respondents are not in Norway. I do not think the fact that I did not meet the respondents in person hugely affected the quality of the answers I got but it is hard to know as I do not have any physical interviews to compare with. During the interview some of the respondents chose not to have their camera on. This was not a problem but this resulted in me not being able to observe them when they answered the questions. There were very little technical issues during the interviews so very little information was lost. All the interviews were conducted in English. The reason for this is that all the information I have read is written in English and it is easier to communicate the topic in the same language. It is hard to say if conversing with the respondents in English was a hinder but none of the respondents seemed to have an issue with this.

Respondents

The respondents interviewed in this study was selected by my company contact. The selection criteria was discussed with my supervisors and the company contact in order to find out who would know the most about technical debt and security debt, i.e. who would be the relevant respondents. According to the company the people with most technical debt knowledge are the architects. As a result, I primarily interviewed architects. An overview of respondents is presented in table 3.2. I was given a list of architects and sent them emails to find out who would be interested in being interviewed. I ended up interviewing 26 people from the same company where all are working in agile development teams. The respondents are from several countries.

Table 3.2 shows the respondents; which roles they have, the size of the team they are a part of, and the duration of the project they are working on. De-

Interview	Respondents' roles	Team size	Project duration
1	SA	10	4
2	SA	20, -	-, -
3	SA	7	4+
4	SA, SE, dev	16	10
5	dev	20	10+
6	SA, SE	15	10+
7	SE, IE	100	20+
8	SA, dev	3, 6	10, 4+
9	SA, team lead	25	11
10	IE, dev	5	5
11	SA, dev	9	1
12	SA, dev	13	5
13	SA, dev	15	1
14	SA, SE, IE	6	3
15	SA, dev	5	10+
16	SA, dev	9	4
17	SA, SE, IE, dev	80	11
18	SA, SE, IE, dev	3	4
19	SA, dev	100	20+
20	SA, dev, team lead	3	8+
21	SA, IE, dev	9	7+
22	SA, IE	17	32+
23	SA, scrum master	28	7
24	SA, SE	10	7+
25	SA, dev, tech lead	100	9
26	SA, SE, dev	6	15+

Table 3.2: The table shows the roles the respondents have, the size of the team they are a part of, and how old the project(s) they are working on is/are. One of the interviews consisted of three respondents.

scriptions of the abbreviations used in the table: SA- service/system architect, SE- security engineer, IE- infrastructure engineer, dev- developer. Some of the respondents are part of more than one team and works on more than one project. This can be seen in the table (e.g. team size 3,6 and project duration

10, 4+). In the thesis I refer to the interviews as respondents, meaning that "respondent 4" is interview number 4.

It is difficult to specify the degree to which all the respondents were relevant given the research problem and the topic of the interviews. As shown in table 3.2, the respondents had a mix of different roles meaning that they had different views on the topic. The respondents have different roles and come from different countries but they all work in the same company.

3.6 Data analysis

After the interviews were conducted, they were transcribed and anonymized. During this process, a document was created for noting the missing information, analytic memos [2] (e.g. "connection between postponed security debt and deliberate technical debt?"), and any distinctive information was pointed out. The missing information that was not gathered during the interviews was requested via follow-up emails. The transcribed interviews were imported into the analysis tool MAXQDA [3]. As the emails were answered, the information in MAXQDA was updated.

Thematic analysis is, according to Cruzes and Dyba, 2011 "*a method for identifying, analyzing, and reporting patterns (themes) within data*" (Cruzes & Dyba, 2011). The synthesis process of thematic analysis starts with all the pages of text and ends up with a number of themes that describe the patterns in the data (Cruzes & Dyba, 2011). Figure 3.2 shows an example of how I was able to go from the interview answers to themes. For analyzing the interviews, codes were used. Miles et al., 2013, p. 71 explain **codes** as "*labels that assign symbolic meaning to the descriptive or inferential information compiled during a study*" (Miles et al., 2013, p. 71). Coding is an analysis method that happen in two cycles; first cycle coding and second cycle coding. One of the elemental methods in coding is called **descriptive coding**. I used descriptive codes when labelling chunks of information, where the labels summarized the information they labelled. (Miles et al., 2013, pp. 71, 74). There are different coding techniques that can be used, e.g. deductive coding and inductive cod-

[2]"*An analytic memo is a brief or extended narrative that documents the researcher's reflections and thinking processes about the data*" (Miles et al., 2013, p. 95)

[3]https://www.maxqda.com

ing. **Deductive coding** is when there is a "start list" of codes before the fieldwork. **Inductive coding** on the other hand is when codes emerge during the collection of data (Miles et al., 2013, p. 81). I used inductive coding as it is quite hard creating codes before the data collection when doing an exploratory case study. Some of the codes I found to be helpful during the data collection was "SD description", "SD process", and "TD description". All codes were described in order for them to be consistently applied over time (Miles et al., 2013, p. 84). I got feedback from my supervisors on the codes and ways to do the analysis. The main themes of the collected data was related to the security debt definition, security debt management, and the relation between security debt and technical debt. After getting some last details from the company contact, I added the analytic memos mentioned earlier to the codes. After adding the memos I looked for relationships between the created codes. To accomplish this, I did **axial coding** in order to not only find general properties of the studied topic but also find different dimensions in the data (Scott & Medaugh, 2017).

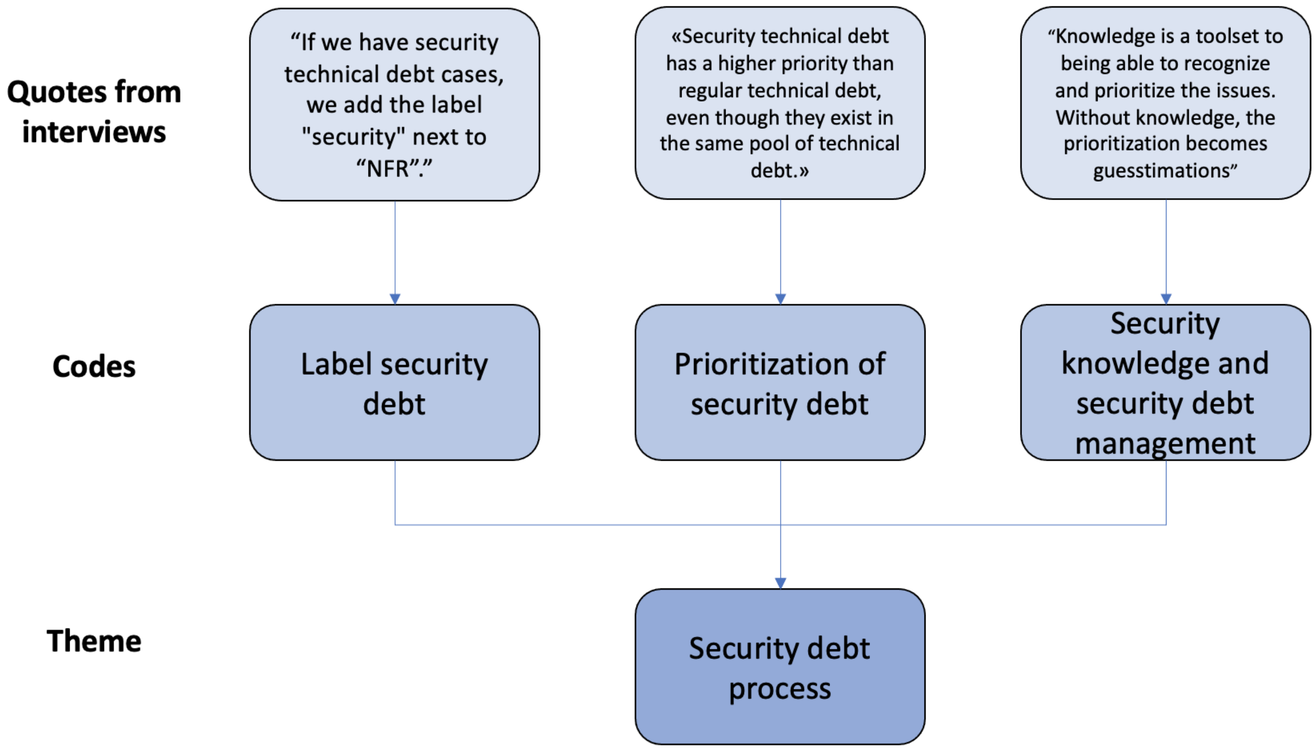

Quotes from interviews
"If we have security technical debt cases, we add the label "security" next to "NFR"."
«Security technical debt has a higher priority than regular technical debt, even though they exist in the same pool of technical debt.»
"Knowledge is a toolset to being able to recognize and prioritize the issues. Without knowledge, the prioritization becomes guesstimations"
Codes
Label security debt
Prioritization of security debt
Security knowledge and security debt management
Theme
Security debt process

Figure 3.2: Example of thematic analysis from interview answers to themes. Here it is shown that three codes are part of the theme "security debt process".

Miles and Huberman, 1994 explain **data reduction** as something that is a part of the analysis. As part of the data reduction, decisions are made regarding the type of data that is relevant for coding and further analysis. A reduction of data have been done as all 230+ pages of transcribed raw data is not relevant when answering the research questions. After the coding and the analysis, the results are described in chapter 4.

3.7 Research ethics

The sensitivity of data in this project is more related to internal company issues rather than personal identifiable information. As the company contact participated in the development of the interview guide, possible security pitfalls were avoided during the interviews. Nevertheless, anonymity was secured both for the respondents and the company.

Runeson and Höst, 2009, p. 146 points out that recording the interview is recommended as it is hard to capture all details if the interviewer only takes notes. I chose to record the interviews for this reason. Before starting the interviews I sent a notification form to NSD [4] (Norwegian centre for research data) because I intended to collect personal data in the form of consent and recording. The notification form needed the interview guide and a consent form. In addition, the storage and the usage of the data that would be collected had to be described and approved before starting the interviews. The first thing that happened before the recording started was that I asked the respondents if they were okay with the recording and that I needed to document their oral consent on the recording. I read the most central points from the consent form, pointed out that I would send it to them after the interview, and then emailed the full consent to the respondents when the interview was finished. In the consent form, a specification was made that the answers they provided would be made anonymous and only used for answering the research problem. This means that all respondents are referred to as "they" and not "he"/"she".

All data from the interviews are stored and treated in accordance with the notification form sent to NSD.

[4] https://www.nsd.no/en

Chapter 4

Results

I will in this chapter present the findings. The information will be viewed together and analyzed. The sections in this chapter corresponds to the research questions described in the introduction (1.2). At the end of each quote I refer to the respective respondents with a number in parenthesis (ref table 3.2).

4.1 Security debt definition (RQ1)

How did the respondents describe the term security debt? A security debt definition will be presented in section 4.1.1 that is a result of the following descriptions provided by the respondents.

From the interviews I found that the respondents have different ways of describing the term security debt. When I asked how they would describe security debt many of them talked about it in more than one way. I chose to create groups of explanations based on the respondents answers in order to see which respondents view security debt in the same manner. Thus, several of the respondents are in more than one group. Some of the groups have a higher number of respondents than others. Table 4.1 shows the created groups.

Three of the groups with the highest number of respondents are 7, 6, and 4. These address **security levels**, the **similarity to the technical debt definition**, and the **postponement of security issues**.

Group number	Security debt explanation	Number of respondents
6	Security debt is related to technical debt	14
4	Security debt is security related issues that has been postponed	12
7	Security debt is described as security levels	10
9	Security debt are known security issues	7
1	Security debt is the needed work in order to address security concerns	6
3	Security debt is having to update third-party dependencies	6
2	Security debt are holes in the system that can be used against you	4
5	Security debt are security cases that do not have straight forward fixes	4
8	Security debt say something about how safe the system is & the overall security footprint	3
10	Security debt are unknown security issues	3

Table 4.1: Groups of security debt explanations based on the respondents' answers and the number of respondents per group. Respondents can be in several groups depending on how they answered.

Groups 7, 2, 1

A respondent from **group 7** described security debt as:

It is clearly an evolution, so you are between two levels of security and you are on the wrong one, on the lower one, so if you are there, you are not allowed to stay there because it's a danger and a risk. (15)

Nine other respondents either described security debt in this way or spoke of examples where this proved to be the case. One of the examples used by several of the respondents is regarding the TLS (Transport Layer Security) protocol version and the need to update. Respondent 8 from group 7 describe the TLS example:

using TLS 1.2 doesn't expose you, doesn't give you a security vulnerability, but as soon as 1.3 is available this occurs because there is a gap. Now you can improve your security by stepping up and using that protocol instead. (8)

Group 2 consists of 4 respondents. They describe security debt as holes that can be used against the organization. The TLS example above explains that an update can close this hole that is a result of the release of a new protocol version. There are also other holes in the system that can be used against the system. Respondent 6 from group 2 explains that if there exists a way to for example kill the performance with the means of a DDoS (Distributed Denial of Service) attack then that existing hole can be maliciously used to get something the attackers shouldn't. This is in close relation to **group 1** where the respondents answered that they view security debt as the work that needs to be done in order to address security concerns. According to respondent 5 from group 1 these concerns can for example be that you have a slow endpoint that can be used in a DoS (Denial of Service) attack. One of the respondents that is in both group 1 and 7 showed the following during the interview:

I'm thinking that we have some sort of quality criteria or imagination of how the system should be kind of: good quality or really secure. So we have our vision here ["right hand raised high"] and we have our state over here ["left hand underneath right hand"] and over here ["the section between the two hand placements"] is the debt. (24)

This description is similar to the description by respondent 15 (quoted above). Respondent 24 showed that there are two levels of security; the vision and the current state, where the section between them is the existing security debt. In order to move from the current state to the vision they have to remove this gap between the two states. This is done by repaying the security debt and/or adding security controls. This gap between the two security levels can be related to both groups 1 and 2 as these groups represent the respondents that view security debt as work they have to do in order to address security concerns/holes that can be used against them. Addressing the work that fixes these issues can be the work needed in order to reach the systems security vision. Looking at the groups 7, 2, and 1 together, the total number of respondents becomes 15.

Group 6

Group 6 contains the 14 respondents that view security debt as a technical debt or describes security debt in a way that points to similarities with the technical debt definition and Fowler's quadrant. The respondents 24 and 10 from this group described security debt as the following:

Technical debt which kind of has a label of security you know. I would imagine it as some improvement pending which should improve security. (24)

... when you know that there are parts in the code that maybe are not optimal when it comes to security and you know that you need to fix it somehow but maybe of course you don't have time to do it well at once so you have kind of have a backlog with the security fixes that needs to be done. (10)

The respondents above described security debt as technical debt that has security implications and as not optimal solutions that impacts the security. The descriptions from the respondents in group 6 can be divided into three sections; not deliberate security debt, deliberate security debt, and security debt due to change in technology. There are respondents that have described security debt in a manner that involve more than one of the sections listed above. Respondent 4 spoke of security debt as both not deliberate and deliberate where not deliberate security debt is a result of lack of knowledge and deliberate security debt is due to trade-offs:

...security debt then it's security due to ignorance or not knowing, not being aware that you have problems. (4)

...when we develop things, when we design things you take shortcuts just to be able to deliver faster and then you are making trade-offs. Security is a good example of such trade-offs. (4)

Another respondent (respondent 17) in group 6 described deliberate security debt as something that is planned and then done. Half of the respondents that described security debt as something deliberate also described it as security levels (from group 7). Respondent 17 also explained that the planning is about choosing security solutions that will help them avoid security vulnerabilities. This respondent join the two groups by connecting security debt as being deliberate choices where these planned choices will get them to a higher security level. The last section is security due to change in technology. Respondent 3 explains that change in context affects the system:

...it might be that it is sufficient now but not in long run, so it's kind of a quick fix so that we get the security measures OK for now, but tomorrow the thing is different because we need to have more strict rules or whatnot, and I would say it's a debt. (3)

This improvement can for example be compared to updating to a newer version of a dependency or doing changes in the code as a result of new technology being created (e.g. updating from Python version 2 to 3). These changes can indicate moving up or keeping the current security level as the context is continuously changing. Respondents have also talked about whether or not security debt can and/or should be kept in certain instances. Respondent 5 said the following:

I don't see a difference between technical debt and security debt [it's all about improvements]. Also, debt is such a negative word, some debt you don't ever need to pay back, it works, but then some things you need to improve because of circumstances (5)

The context can change as explained by respondent 3's earlier. The security solutions can be sufficient for now but things are different in the future which means that improvement must be done. Respondent 5 also referred to security as a factor for attracting attention to the issues:

security is just a way to maybe measure how important or justify how important the fixes are, how much attention the thing needs. (5)

On the other side respondent 6 disagrees with what has previously been said regarding not repaying security debt. Respondent 6 mention that the security perspective is always important and security debt is something they do not choose to keep:

...we cannot live with that kind of changes or features that are not kind of secure enough. So we are not really taking shortcuts in that area while in technical debt side we might take shortcuts. (6)

Here is a clear difference between security debt and technical debt according to respondent 6. Security debt, according to the same respondent, is not something you take on deliberately because there are no shortcuts in security, but rather something that you do not know of:

... we are not kind of cutting the corners and accepting some security issues, so that's not happening. But it doesn't mean that we don't have security issues, of course we have, people are writing the code so we have security issues... (6)

Group 4

The last group that contains a high number of respondents is **group 4**. This group contain the 12 respondents that view security debt as something that can be postponed. This is explained by respondent 7:

... that's more security debt in that we have accepted the risk and while we might not do something about it now you might do something about it in the future. (7)

This description indicate a connection to the security debt process, meaning that the respondents in group 4 associate security debt with the process of postponement. What is interesting is the reasons behind the postponements. This is something that will be further discussed in chapter 4.2. It was explained earlier that many respondents view security debt in more than one way and are therefore in more than one group. Only two of the respondents in group 4 is not found in group 7 and/or 6.

4.1.1 Definition

The groups with the most respondents have been explained. These groups of respondents explain security debt as:

- Security levels
- The similarity to the technical debt definition
- Due to postponement of security related issues

In addition to describing each group individually they were also looked at together in order to see similarities. **Group 7** was discussed together with **groups 1 and 2**. They pointed out that security debt can be the gap between two levels of security where the gap is the work that needs to be addressed in order to reach the top level of security needed for the system. The respondents from **group 6** described security debt in three ways:

- not deliberate security debt
- deliberate security debt
- security debt due to change in technology

One of the respondents from the group connected deliberate security debt and security levels with the help of planning. It was also explained that keeping up with the change in technology could indicate moving up or keeping the current security level. Here it is shown that there is a connection between groups 7 and 6. **Group 6** also have two respondents that had different views on whether or not security debt in some circumstances can and/or should be kept just like with technical debt. The last group, **group 4**, was briefly explained because of the high number of respondents. They associated security debt with the process of postponement. Because postponement is a process action it did not have a direct impact on the formulation of the security debt definition. But as almost all of the respondents in group 4 was also found in group 7 and/or 6 it means that many of the respondents from the other large groups also view security debt as something that is not fixed immediately due to various reasons.

Following the discussion above I propose the following definition of security debt:

Security debt is a set of design or implementation solutions that hinder or has the potential to hinder the achievement of a system's optimal/desired/required security goal

---

Security debt definition explanation

The security debt definition does not limit security debt to being one specific type of problem. It is described in a way that shows that security debt is related to the fact that there exist a better solution to an issue. These non-optimal solutions are a collection that makes it hard or has the potential to make it hard to get the optimal/desired/required security for a system. Not all design or implementation solutions necessarily hinder the achievement of the system's optimal/desired/required security goal, but rather has the potential do hinder. What is meant by this is that it is possible to have security solutions that are less than optimal/desired/required but the security solution does not make out any danger as long as it is an internal issue. The team knows about the security issue but chooses not to fix it while it is internal. As soon as the system is exposed to environments where they can be misused , the security issue that did not pose any danger earlier has now become security debt. The security in a system is dependent on the context. Some respondents mention that they do not compromise security in any circumstances (e.g. respondent 6) while others (e.g. respondent 5) might have situations where security suffers from the trade-offs.

The use of optimal, desired, and required as ways of describing the needed security in a system allows for people from different contexts to choose what fits them. The use of goal in the definition refer to the system's security vision described earlier in the section. The use of security goal incorporate security levels and security criteria as both these point to an optimal/desired/required security. When explaining the respondent's most favoured views on security debt it was mentioned how security debt can occur in more than one way, e.g. as both deliberate and not deliberate. This is not mentioned in the definition but there is no limitation pointing to how security debt is accumulated.

Some of the groups created in order to categorize the respondent's answers did not have a direct impact on the formulation of the security debt definition as they consisted of few respondents and did not have a direct connection to the larger groups. Even though not all groups was used in the formulation of the definition, it does not exclude them. The groups 3, 5, 8, 9, and 10 contain the respondents that view security debt as having to update third-party dependencies, security issues that do not have straight forward fixes, it says something about how safe a system is (overall security footprint), known security issues, and unknown security issues.

Groups 8, 9, 10

Group 8 contain the respondents that view security debt as how safe the service is and the overall security footprint. Respondent 22 describe the overall security footprint as:

> *... all the things that are lying in the system, either that are discovered and put on the board or not.* (22)

The reason for this is that there is no secure system, meaning that there are probably some things that cannot be found by tools or by the people working on the system. This divides group 8 into security debt as known (**group 9**) and unknown (**group 10**) security issues, where respondent 22 is one of the respondents that is in both groups. These groups contain the respondents that specifically mentioned whether or not the security issues were known or not.

Groups 3, 5

The last two groups are 3 and 5. **Group 3** describe security debt as the need to do updates (e.g. TLS example from earlier) and **group 5** explain that security debt cases are not straight forward, meaning that they are not easy to fix. Respondent 26 from group 5 points out that security debt is the

...overall problem in the end, it's not particularly related to a certain version of whatever component, it's the overall picture (26)

This is in contradiction with group 3 where security debt is described as the need to do updates. The same respondent also explains that in cases where for example a package update in order to fix a security vulnerability is not possible because of a deadlock problem, this will be a security debt because larger changes needs to be made in order to fix this part of the system.

4.2 Difference between security debt and security vulnerabilities (RQ1.1)

In this section I will look at how the respondents view the difference between security debt and security vulnerabilities. Their answers resulted in the creation of several groups, just like for the security debt definition. As with the security debt definition groups, several of the respondents are in more than one group. Table 4.2 shows the groups.

When asked how security debt is different from security vulnerabilities 16 respondents said that security vulnerabilities is like a current status, it is something exploitable that should be fixed fast. Respondent 11 explained the following:

a security vulnerability outcome to something that can be exploited with knowledge about the vulnerability. (11)

Group 1 is the group with the highest number of agreeing respondents. They described that the difference between security debt and security vulnerabilities is that security debt is something that is postponed while security vulnerabilities will not be postponed. It is the reason behind the postponement that is interesting as the postponement itself is part of the process, it is a reaction.

Group number	Difference between security debt and security vulnerabilities	Number of respondents
1	A security vulnerability is a current status, it is exploitable, and security debt is something that has been postponed	10
3	Security debt is the same as security vulnerabilities	9
6	Security vulnerabilities is a part of security debt, a subset	8
5	Postponed security vulnerabilities become security debt	5
2	Security vulnerabilities are easy to fix and security debt is not easy to fix	2
4	Security debt can lead to security vulnerabilities	1

Table 4.2: Groups that explain the differences between security debt and security vulnerabilities based on the respondents' answers and the number of respondents per group.

Group 1 is divided in four as a result of there being four mentioned reasons behind the postponement. Respondent 10, 15, and 18 described it and 25 gave an example:

a vulnerability is just a weakness in the system I guess, but security debt is something you knew about, that you are aware of but you have not had time to fix it yet. (10)

it's the danger or the risk that comes that affect us. I mean you feel the risk because you are responsible ... Normally the attitude in the team is that if it's security related you don't postpone it but sometimes, as I told you, you cannot address it if you have dependencies or something, so then it's becoming a security debt. (15)

a vulnerability is something like concrete, this is vulnerable, this can be exploited, it has to be fixed, it has bigger priority. And security technical debt is something we need to change in our security or something which is not vulnerable right now but fixing those things would make things better technically. (18)

[Example about change in encryption] I would say that the risk that someone is able to hack this encryption is much less than the risk that we'll create some

bugs with this migration or conversion to the new encryption method. This is technical debt or security debt in my opinion. But we have evaluated the risk to be very low and the risk by migration or doing something with that is higher than not doing that. So this is an example with low priority. (25)

What can be seen here is that the respondents explain that the postponement is due to not having enough time, having dependencies that makes it very hard to repay the security debt, it can be seen as strategy in order to become more secure, and that some issues are so low risk that it is not worth fixing yet. Almost all of the respondents that view security debt as something that can be postponed also said that security vulnerabilities have a higher priority than security debt and will therefore be addressed first. Security debt and security vulnerabilities can also be described as being the same. The 9 respondents that agree with this is found in **group 3**. Respondent 21 explained the following:

I would say it is the same ... because both can be vulnerability and debt. They can point to there is some something existing weakness in the service. They can also be prioritized in different ways, something we can live with, something we will fix, something that is critical. (21)

Here it is described that security debt and security vulnerabilities points to weaknesses in the system and that they are not really different. In addition to talking about security debt as something that is postponed due to dependencies, explains respondent 15 from the same group that differentiating between security debt and security vulnerabilities is a philosophical problem. They do not really do that. This is due to the fact that they are responsible and they feel the risk that can affect them. Respondent 5 agree with this. They do not differentiate between anything; they view everything as equal, meaning that it is the evaluation and the context that decide what they do next. Many other respondents agree that it comes down to the analysis of the issues of whether or not they are repaid. The third largest group is **group 6**. The 8 respondents in this group view security vulnerabilities as a subset of security debt. Respondent 8 and 19 explain the following:

I would say that security vulnerabilities is like a subset of all these issues [security debt]. This is like the ones that is really severe that you should fix tomorrow, or today if possible. And then there is this superset which is kind of

OK there is an issue there but it's not a fire, it's not serious, you should fix this. (8)

I would say security debt includes security vulnerabilities. But there is more to it. So I would say security vulnerabilities is one of the inputs into the security debt and I would say there are different inputs in this security debt. I would include also those unknown- we need to find as well. (19)

The two respondents above point out that security debt is this larger set where security vulnerabilities is a subset. Respondent 19 further explains that security debt also has other inputs where one of those inputs are issues that are yet to be found. This respondent is also in the two groups that view security debt as both known and unknown security issues from section 4.1 shown in table 4.1. Another respondent that pointed out that security debt is more than security vulnerabilities that are not fixed is respondent 24. They pointed out that lack of tooling (e.g. static code analysis tools) and intrusion detection systems are things that are considered security debt as well. The reason for this is that it is measures for trying to counter upcoming issues like vulnerabilities. Another group that is a bit smaller than group 6 is **group 5**. This group view security debt as the postponement of security vulnerabilities. This is explained by respondent 23:

... if there is some vulnerability for some reason doesn't have priority. If we don't do anything then eventually it will be debt of course if it's postponed. (23)

This group separates from group 1, the group that was first mentioned, because in this case the security vulnerability is postponed due to low priority. Respondent 20 point out that when a security vulnerability becomes a security debt it is still considered a security vulnerability but it is also considered a security debt. The repayment of this debt will also resolve the security vulnerability. On the other hand respondent 16 from **group 4** explained that a security vulnerability can be a result of a security debt and not the other way around as described by group 6. Respondent 16 explains:

If the security vulnerability is caused by security debt because of lack of upgrading library then upgrading the library will also fix the security vulnerability. (16)

Respondent 16 and 20 both agree that repaying the security debt will also fix the security vulnerability. **Group 2** is the last group and consists of respondents that view security vulnerabilities as something that might have easier fixes than security debt. Respondent 1 gave the following explanation:

Vulnerabilities, sometimes I see them as easier to fix if the fix will be like I just need to upgrade to a new version of that package or to add an extra input validation that is not done by a given library. This is how to solve a vulnerability issue. When it comes to security debt I think sometimes that might involve a bit more on refactoring. It is not something quick to do. (1)

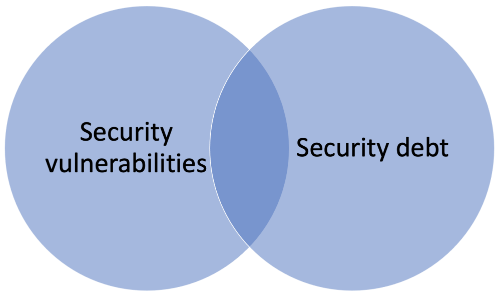
Respondent 26 agrees with this and described that security debt would in certain situations result in larger changes, for example if there are deadlock problems that makes it impossible to do the wanted updates in order to keep the system up to date.

Relation between security debt and security vulnerabilities
The relation between security debt and security vulnerabilities has been described by the respondents in seven different ways. The groups with the largest amount of respondents are 1, 3, 5, and 6. Together they make up 24 of the respondents, where five of the respondents overlap, meaning that 19 respondents only appeared in one of the four groups. Two figures (4.1) have been created in order to visualize the relation between security debt and security vulnerabilities. Figure 4.1a show that there are some security issues that are both security vulnerabilities and security debt while other issues are only in one of the groups and figure 4.1b show that security vulnerabilities is a subset of security debt.

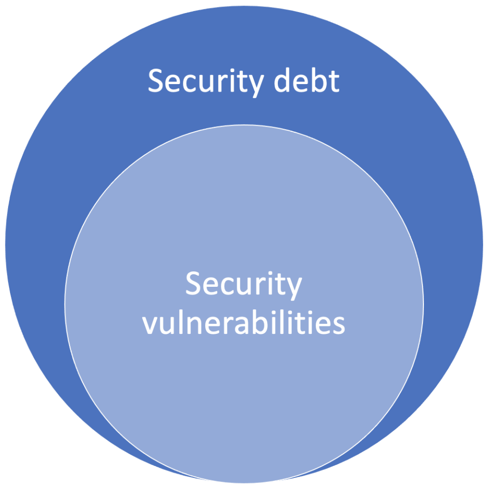
At the start of the section it was explained that security debt as postponement of security issues could be divided into four reasons;

1) time limitation,
2) dependencies that make it hard to repay,
3) as a strategy for more secure systems, and
4) low risk issues not worth fixing.

These are all types of security debts that can be repaid as better solutions exist, they might not be easy to implement, but they exist. Respondent 9 on the other hand did not describe security debt as postponement but said that



(a)



(b)

Figure 4.1: Two visual representations of the relation between security debt and security vulnerabilities. Figure (a) have a shared area between security debt and security vulnerabilities while figure (b) shows that security debt completely surrounds security vulnerabilities.

a typical security debt that will not be prioritized for repayment is when a dependency or a library have a security issue but there is no available updates. It is then not possible to fix the security issue because of the lack of a solution. This goes against the definition of security debt, if there is no better solution then it cannot be a security debt as security debt states that a better solution exist. This type of security issue is then not a security debt but rather just a security issue that cannot be solved. The relation between security debt and security vulnerabilities can be further described because of this. Security vulnerabilities is according to respondent 15 something that just pops up. If the security vulnerability has a solution then it is a security debt until it is fixed because a better solution exists. On the other hand, if the security vulnerability does not have a possible fix then it cannot be a security debt because of the lack of a better solution. As soon as it is possible to fix the security vulnerability it becomes a security debt until it is fixed. Respondent 24 explains that security vulnerabilities are explicit things where as soon as the team is aware of them and see that there is a need for improvement, they are considered to be security debt. Many security vulnerabilities are short-term security debts as they often are fixed fast. Respondent 7 explained that security debt can be divided into short-term debt and long-term debt:

... until they [security issues] are fixed there is a cost and the cost is that we are vulnerable so in a sense they are a security debt until they are fixed but

they are not a long-term debt, it is just more like a credit card compared to a house loan. (7)

The two figures 4.1a and 4.1b propose relations between security debt and security vulnerabilities. There are three sides to figure 4.1a:

1) Security vulnerabilities with existing solutions that are not fixed (postponed) is security debt.
2) Security vulnerabilities that do not have existing solutions are not security debt.
3) Other security issues that are not security vulnerabilities can also be security debt.

Figure 4.1b show another way of displaying the relation between security vulnerabilities and security debt. Here **security debt completely enclose the security vulnerabilities**. Both figures 4.1a and 4.1b show that **security debt can be something other than only security vulnerabilities** but figure 4.1b show that **all security vulnerabilities are also security debt**.

4.3 Security debt process (RQ2)

The company has several processes for different parts of the development of their software products. One of the processes is for the management of technical debt. Respondent 22 briefly describe the technical debt process and its importance:

... *generally I'm putting, or the developers are putting, some NFR technical debt in the Jira board, say let's refactor code, an annotation, fix me or /TODO, or something like that. So that you don't forget about it. The problem with this is that if you don't make clear cases and don't have a process it will just stay there, nobody will really look at it after.* (22)

The quick description from respondent 22 of the technical debt process will be described more in depth in this section. During the interviews I asked the respondents how they think the security debt process should be. The answer with the highest amount of respondents was regarding the use of their already existing technical debt process. Respondent 3 said the following about the process for security debt:

... *these [security debt] should be prioritized higher than the technical debt issues, but otherwise, the process should be pretty much the same.* (3)

Even though not all respondents thought that the technical debt process would work for security debt, all respondents were asked the same type of questions about security debt as with technical debt. The official company process for technical debt is comprised of several sections: prevention, identification, documentation, analysis, monitoring, communicating, planning, and repayment. Each section describe a part of the technical debt management. I will describe all sections but the main focus will be on the section that is mostly mentioned by the respondents.

Technical debt self-assessment
Each year the teams perform a self-assessment of how they work with technical debt. The sections in the process is divided into levels based on a maturity model used in the company. The maturity model and the technical debt process was described in a meeting with a company employee and by an internal company document. When the teams assess their technical debt work they find what level they are on for each section. The levels are:

Level 1 - unorganized
Level 2 - semi-organized
Level 3 - organized
Level 4 - super organized

The different levels says something about the amount of work they put in for technical debt. Level 1 is the lowest level where the team does not work with technical debt and level 4 is the highest possible level where they have a structured and more strict way of working with technical debt. The target level for the teams is the organized level (level 3).

Figure 4.2 gives a simple overview of the flow of the technical debt process and the annual self-assessment they perform. Technical debt can be prevented after its identification or it can be documented, analyzed, planned, and then repaid. The sections monitoring and communication are continuous during the technical debt process. I will go more into depth on the different sections of the technical debt process below.

Each team work with technical debt differently. Some teams are quite strict

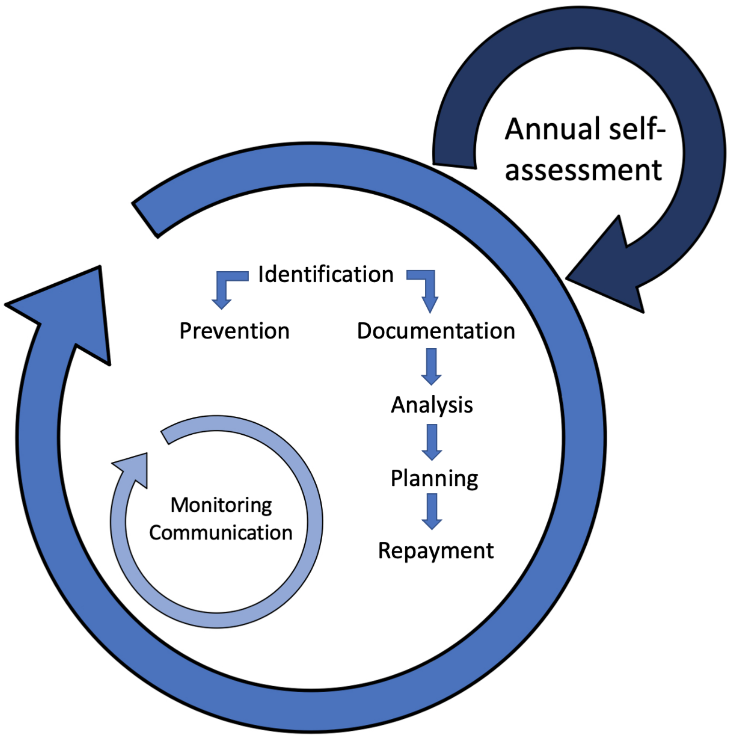


Figure 4.2: A simple visual representation of the technical debt process flow and the annual self-assessment for finding how the team currently work with technical debt.

while others tend not to be as strict. Most of the respondents talk about the technical debt process in a way that matches one of the levels for each section. It can be argued that level 1 in fact is not a process seeing as the teams then do not work with technical debt. They then have the potential for improvement especially since the target level is the organized level. If they are not on level 3 or higher, they can be seen as "deviating" from the process seeing as how they are meant to be on level 3. In this case they have to register a ticket in their management tool, Jira, that says that they are not on level 3 and this is added as something they have to work on.

Table 4.3 gives an overview of the level 3 approaches from the technical debt process that can be applied to the management of security debt. The bold text mentions additional approaches that can be used in the security debt process. The sections, approaches, tools, etc. will be explained underneath.

Section	Approaches
Prevention	The team follows coding standards and performs code review for all non-trivial changes
	Threat modelling
Identification	The team identifies security debt during implementation and code reviews
	The team identifies security debt using static code analysis tools (e.g. ReSharper, SonarQube, Coverity)
	The team identifies security debt as part of operational processes like incident reviews and problem management
	Security debt is identified through a bug bounty program
	Security debt is identified by tests performed by the internal security team in the company
Documentation	Security debt is documented in the same backlog as everything else
Analysis	Security debt is analyzed and a severity score is calculated and documented in Jira
Monitoring	Different types of security debt metrics is measured over time by looking at the sum of all severity scores and metrics in SonarQube
Communication	The team continuously communicates security debt risks to stakeholders, focusing on the business and/or customer value of paying down the security debt with the highest severity score
Planning	The team decides on what security debt should be paid down next based on the severity scores
Repayment	Some of the team's capacity is always reserved for paying down security debt (20% or more is the industry standard and highly recommended for technical debt)

Table 4.3: Security debt process adapted from the studied company's technical debt process. The sections are presented together with their corresponding approaches. The additional approaches that are specific to the security debt process are highlighted in bold.

4.3.1 Prevention

The first presented section, prevention, is about to what extent the teams follow coding standards and the code review activity in order to prevent the accumulation of technical debt. 21 respondents talked about code reviews as a means for preventing technical debt. Among these respondents there is a range from "not important" to "important" as described by respondent 5 and 9:

We do not enforce anything basically, so it is up to the person or the author itself or the authors to get the feedback on the code produced. (5)

We do code reviews on all code (including infrastructure code), and its not allowed to push to our main branch without a code review. (9)

The first code review description matches level 2 and the second points to level 3. Finding technical debt during code reviews is a good way of preventing them. It gives the team the opportunity to either fix it before the merging of branches or to create a ticket in their management tool. The use of coding standards are also mentioned. Respondent 5 explained that following coding standards is a central part of the technical debt process.

The use of code reviews is also mentioned by the respondents as an approach to prevent the accumulation of security debt. Having the opportunity to fix the security debt before it is materialized in the system is a good way to prevent potential misuse of the system because of the security debt. It follows from these arguments that security issues such as security vulnerabilities found during the code review and other preventative approaches before the merging of branches is not considered security debt as they are not yet materialized in the production code. Even though 21 respondents mentioned code reviews for technical debt only four specifically mentioned it for security debt. The reason for this might be that security debt is a term that has not been researched very much and might therefore not be something the respondents and their teams specifically look out for. Respondent 1 was the only one that mentioned **threat modelling** for avoiding security debt. This happens when there is a change in the architecture as these changes can introduce security debt.

A summary of the key points:

- 21 respondents spoke of code reviews as a means for preventing technical debt where the importance of the code reviews range from "not important" to "important".
- Four respondents specifically mentioned code reviews for preventing security debt. The decrease in numbers from technical debt might be due

to security debt not have been researched as much as general technical debt.
- Threat modelling have been mentioned as an approach to help prevent security debt.

4.3.2 Identification

This section focuses on how the teams identify technical debt. Level 3 mention identifying technical debt during implementation, code reviews, by using static analysis tools (e.g. SonarQube, ReSharper, Coverity), and as part of operational processes. All 26 of the respondents mentioned using various tools in order to identify technical debt where the most mentioned being SonarQube (21 respondents), Coverity (22 respondents), and Snyk (22 respondents). Not all respondents work on projects where the use of the standard static code analysis tools can be applied, but in those cases other tools are used (e.g. linters). In addition to using tools, almost all respondents said that they find technical debt manually, meaning that technical debt is identified not only when they are performing code reviews but also when they are working on the code, for example when implementing. One of the respondents explained it nicely with an example:

... boy scout- leave the camp area a nicer place than it was when you arrived. (14)

When explaining this the respondent meant being on the lookout for issues and other problems such as technical debt that they could fix or notify so that the area in which they are working will be better then when they started working there. Respondent 5 on the other hand is one of the respondents that do not actively look for technical debt because as long as the system works it does not need fixing. Respondents have also mentioned that they have testing as part of their pipeline and that this is also a means for identifying technical debt. Respondent 2 pointed out that technical debt could also be identified during incidents. The majority of the respondents described the identification of technical debt that matches the third level. The security debt process differ a bit from the technical debt process here. When it comes to identification 11 respondents pointed out that security debt could be found manually, where some mentioned code reviews and others mentioned that they could identify security when going through or working on the code. On the other hand two respondents said that finding security debt manually in this manner is quite difficult (this will be further discussed in section 4.5). One of the differences that sets the identification of technical debt and security debt apart is the use of **bug bounties**. 16 respondents said that being part of the bug bounty program helps them identify problems in their system. Respondent 5 and 4 said that

> *... this bug bounty has maybe been the most valuable thing when it comes to finding security issues and like people actually having the time to look at solutions and try to find loopholes. That has been super-valuable.* (5)

> *I see very big value in this this bug bounty program, because it's like when we're hiring external security guys who tell us- look in your code you have this problem, and yes this problem might come from using third-party library which brings the problem, but it's also from your code.* (4)

This bug bounty program reports all kinds of different security issues that can be problematic in the systems. Some of these issues can be security debt while others are not. In addition to the bug bounty all 26 respondents have tools that run periodically as mentioned above. The tools SonarQube, Coverity, and Snyk are the tools that are mentioned the most and these can detect security issues among other things. In addition to this mentioned 4 respondents that **security people inside the company manually tests** their system as these people know how to identify specific issues. Three of the four respondents that mentioned security testing specifically pointed out **penetrations testing** for finding security issues. These security issues might be security debt but other security issues may be found as well.

A summary of the key points:

- Technical debt is identified manually (when implementing and during code reviews), with the use of tools (e.g. SonarQube, Coverity, Snyk), during testing, and during incidents.
- 11 respondents spoke of identifying security debt manually, where some mentioned code reviews and others mentioned finding security debt when working on the code.
- The same tools used for identifying technical debt are also run for finding security issues.

- The bug bounty program can be used for identifying security issues, including security debt.
- Four respondents mentioned testing for finding security debt, where three of the respondents specifically mentioned penetration testing.

4.3.3 Documentation

This sections is regarding the documentation. Level 3 in the technical debt assessment describe that technical debt is to be documented as everything else: in the same backlog. The majority of the respondents spoke of having everything in the same backlog as described by respondent 17:

... when it comes to technical debt we are having technical debt tasks in the same backlog as the features. They are prioritized together and so on. (17)

Respondent 5 is one of the respondents that do not keep technical debt in the same backlog as everything else, but pointed out that having different backlogs for different things is not helpful for them. The reason for this is that it is the same people working on everything, it is the teams responsibility to work on the important things together. 25 respondents said that they are adding technical debt tickets in Jira but it does not mean that all technical debt that is identified is registered. Two of the respondents that mentioned this is respondent 11 and 14. Respondent 11 explains that they only add it to Jira if there is a probability that they will fix it and the reason for that is that they do not have a process for technical debt in place yet (as they are not part of a formal company <process> at this time). Respondent 14 explains that if a tool finds a technical debt then they will probably just fix it while working and not create a Jira issue as it is often simple things. On the other side, respondent 22 said:

I'm adding it [technical debt] to Jira because it gives me leverage in discussion with management. Although I could fix things quick and dirty and just make it work. The idea is that I can have a leverage to explain. (22)

This respondent points out that even if there is a technical debt that is a quick fix it will still be added to Jira as it is a means of keeping track of the time spent and the issues that they have.

Labelling technical debt and security debt

Labelling the technical debt registered in Jira makes it possible to filter them so that an overview of the technical debt can be shown. The yearly assessment that the teams need to do states that the technical debt should be labelled. 24 respondents mentioned the labelling of technical debt but not all respondents label the technical debt every single time they register a new ticket. Respondent 10 mention that they are more focused on the use of labels when they are doing the assessment. Several labels can be attached to the tickets registered in Jira. Most of the respondents talked about labelling the technical debt with the "NFR" label (non-functional requirement). In addition, labels that describe the source of the identified technical debt such as "Coverity" and "assessment" can be added. Respondent 2 explained the following:

> *In Jira, we tag technical debt cases with "NFR" (non-functional requirements). This may change at some point. If we have security technical debt cases, we add the label "security" next to "NFR".* (2)

The security debt process is not very different from the technical debt process when it comes to documentation. As described above, respondent 2 mentions that an additional label, "security", is added together with "NFR". Almost all respondents that mentioned using "security" also mentioned adding the "NFR" label. Respondent 14 said the following:

> *Security typically takes precedence right, so anything related to security typically gets the "security" label and then it [security debt] may also get the "technical debt" or "NFR" label.* (14)

Respondent 14 points out that using the "security" label helps show the importance of the security debt. As with technical debt, other labels can be added, for example its source and if it is detected during an assessment. Even though most of the respondents use some labels when adding a security debt ticket in Jira, not all have a specific/strict way of labelling this specific debt. Respondent 3 explains the following:

> *Not for now [labelling security debt] but I think it's a good idea. Something that I haven't really thought of before.* (3)

Just like with technical debt is security debt mentioned to be in the same backlog as everything else.

A summary of the key points:

- Technical debt and security debt is documented in the same backlog as everything else.
- Technical debt is often labelled with "NFR" (non-functional requirement). Additionally, labels that specify the source of the technical debt can be added.
- Security debt if often labelled with "NFR" and "security". Additionally, labels that specify the source of the security debt can be added.

4.3.4 Analysis

In this section the technical debt is analysed. Level 2 mention that the teams use the priority field (low, medium, high, critical) in Jira while Level 3 is more in depth and specify the calculation of the severity of the technical debt, where the severity is calculated as a product of impact (what is the damage/consequence to the system/organization?) and likelihood (what is the probability that the debt will have an impact on the time horizon of the product roadmap?), $S = I * L$. The severity, impact, and likelihood is referred to as "risk severity", "risk impact", and "risk likelihood" in the documentation. Applying this analysis to the technical debt tickets says something about how severe/critical an issue is.

It is hard to distinguish between the use of priority and severity as the respondents talk about it in a mix, but 18 respondents mentioned severity and 6 talked about using the priority field. There are a few that specifically mention the difference between using the severity score and the priority field. Two of these respondents are number 9 and 10. They said the following:

I'm in the process actually now to convert [from using priority to severity], or to make the correct severity and impact on these things... Moving issue by issue over to that. (9)

... in our daily work we don't do this [calculate severity]. We just set the priority of the Jira issues based on our experience of how great the security risk would be if we don't fix the issue. (10)

Putting the severity vary from important to something that the respondents do not spend time on. Respondent 9 described moving from priority to severity. This indicate that they are working on moving from level 2 to level 3 in the assessment. Respondent 10 on the other hand do not focus on the calculation of the severity but rather uses the priority field. A few respondents talked about not being strict on adding analysis to the technical debt Jira tickets at all. Respondent 3 said the following about severity, impact, and likelihood:

Yes, that is kind of how it is set in the paper but in the real life it is most likely that we work it immediately when we find the problem. It quite often goes like that- we fix it. (3)

This respondent explains that they are not strict on calculating and adding severity. Two other respondents (18, 20) pointed out that they do not calculate the severity every time they add a new technical debt ticket. Both respondent 18 and 20 said that they only calculate the severity score once a year. Respondent 18 calculate the overall severity of all the technical debt cases while respondent 20 calculate for each individual technical debt case (because of the yearly assessment). Respondent 20 also explained that they discuss the prioritization of the technical debt together with the service architect and the business owner. A strict calculation of the severity score is therefore not something they do.

As with technical debt, it is not easy to differentiate between the use of severity and priority in the analysis of security debt. It stands to reason that the respondents use the same analysis method as for technical debt, only that the importance is considered to be higher in the case of security debt. Respondent 18 (from above) is an exception and pointed out that they only add individual analysis scores to security issues identified by tools such as Coverity and Snyk. Respondent 11 is also one of the respondents that do not strictly follow the analysis guidelines. This respondent include the team size as a factor when it comes to the calculation of the severity:

If you are a small enough team then you might have some sense of the respective priority without turning to too many matrices and loaded parameters for your calculation. (11)

18 respondents said that they calculate the severity based on their experience and 6 respondents said that if the debt is reported by a tool then the score from the tool is taken under consideration. In addition, respondents 16

and 8 mentioned that they do the calculations together in the team during one of their meetings. 7 respondents specifically mentioned that they have scales in the team for determining the likelihood and impact. The scales are determined by the team and vary from 1-3 to 1-10.

A summary of the key points:

- The technical debt and security debt can be analysed by the use of a priority field or with a severity score (where the severity score is the product of impact and likelihood).
- Adding the severity score to the technical debt varies from important to something the respondents do not spend time on.
- Security debt is considered to be more important (have higher priority) than technical debt.
- 18 respondents said that the severity score is calculated based on their experience and 6 respondents said that when the debt is found by a tool then they take that analysis under consideration.

4.3.5 Monitoring

The monitoring section is about keeping an eye on the amount of technical debt and using different kinds of metrics in order to control the amount of technical debt. Monitoring was not specifically mentioned by the respondents when it comes to technical debt or security debt. Two exceptions to this include respondent 18 and 21. Respondent 18, as explained in the last section, go through all their technical debt once a year and calculate the overall severity. They then have the opportunity to compare the yearly results in order to see if the technical debt and security debt is more or less severe than previous years. The other exception is respondent 21. This respondent talks about having a special routine each week that is related to both security and technical debt. During this routine they go through all their monitoring, the indexes, their scanners, etc. in order to get an idea of how "well" the system is.

A summary of the key points:

- Few respondents spoke of monitoring during the interviews.
- Respondent 18 explained that they calculate an overall severity score of all their debt once a year. This score can be used to compare the

yearly results in order to see if the severity is increasing or decreasing compared to previous years.

- Respondent 21 explained that they have weekly check-ups for checking how "well" the system is.

4.3.6 Communication

Communication is about to what extent the team talk about technical debt internally (level 2) and how much they talk about technical debt to the outside, e.g. stakeholders (level 3). 7 respondents explicitly spoke of the importance of talking to stakeholders when it comes to technical debt. Respondent 25 explained that

... we do also involve stakeholders in that [technical debt] process and prioritization because it's also, you know, it has a cost because it takes a lot of resources. (25)

It is also mentioned by some respondents that they talk about technical debt inside the team on a regular basis. Some have specific days during the week/month where they have extra focus on technical debt while others have meetings dedicated to technical debt. During these meetings they might discuss, analyze, and prioritize them.

Security debt is also something that is communicated both inside and outside the team. This is something 8 respondents mention. Respondent 1 is one of them and points out the importance of communicating the security debt to the interested parties (e.g. stakeholders):

Well, I think they [security debt] need to be visualized or, not only visualized, but they need to, everyone needs to be aware of them. Technical debt usually it's inside the team so and the team has an overview of them and these are not something that are presented to stakeholders or because it's sometimes is very specific and they will not understand it. For security debt I think it should be better presented outside of the team so to the ones interested like stakeholders because a security debt has an impact on everyone, and I think it's easier to explain it and the need for it than the regular technical debt (1)

Security debt is also something that is discussed inside the team for example when it is time for backlog prioritization.

A summary of the key points:

- 7 respondents spoke of the importance of communicating the technical debt to the stakeholders.
- Some respondents mentioned that technical debt is communicated inside the teams on a regular basis.
- 8 respondents mentioned the importance of communicating security debt, not only inside the team but also outside the team (include the stakeholders).

4.3.7 Planning

This section is connected to the analysis. Level 2 describes the planning of technical debt repayment based on the priority field and level 3 says that the repayment is based on the calculated severity score. 5 respondents specifically said that for larger technical debts that cannot be repaid in a limited time frame are added to their road map that usually span over 1 year. 5 other respondents explicitly mentioned backlog grooming when planning the next period of time. Respondent 16 said the following:

... *when doing the backlog grooming together with the service owner we decide to prioritize those with higher severity before those with low.* (16)

Although only 5 respondents mentioned the term backlog grooming, said 18 of the respondents (including the 5) that they have meetings where they discuss the backlog prioritization. The prioritization of technical debt is according to 22 of the respondent either based on the severity or the priority set for the technical debt, meaning that some of the respondents are on level 2 and others are on level 3. Respondent 22 on the other hand described the prioritization of technical debt a bit differently:

we are considering the severity as being risk and it's a product between likelihood and impact. But not always you start with the most impactful one, sometimes you just go in a greedy manner because you know that some things are easy to fix and you construct a scaffold for fixing the others in time. (22)

Here it is shown that even tough most of the respondents follow the prioritization done during the backlog grooming, there are others that have other ways of doing it.

Just like for technical debt is security debt prioritized based on the priority and the severity. Security debt is, as explained above, something that gets a higher prioritization as a result of higher importance. Respondent 1 points out that having security focused team members and a small team helps the prioritization and repayment of security debt:

I do not know if it is all the teams, but being a smaller team and having a PO [product owner] that is always focused on security and wants us to make sure that we are good in that part, so we can just start when we see that there is something burning, then we will not wait. This does not happen everywhere.
(1)

The prioritization of security debt is something I will be diving more into in section 4.4.

A summary of the key points:

- Larger technical debt items that cannot be repaid within a reasonable time frame is added to the road map.
- Technical debt is usually prioritized based on the given severity score or the given priority according to 22 respondents.
- Security debt is found to be prioritized based on the given severity score or the given priority field.

4.3.8 Repayment

The last section is repayment. Level 2 specifies that technical debt is sufficiently repaid and level 3 says that the repayment of technical debt should be reserved for a portion of the team's capacity, where 20% capacity is the industry standard. In this section there is also a fourth level, superorganized. This level is for the teams that have a cap/threshold of maximum technical debt that they can have. When this cap is reached or exceeded, 100% of the teams capacity is used for repaying the technical debt.

Of the 18 people that mentioned that they have recurring meetings for the backlog prioritization spoke only 4 respondents of percentages for technical debt repayment, but they do not necessarily follow the percentage. In addition to this mentioned three respondents that **penalty points** is a motivational factor for technical debt repayment. Respondent 6 said the following:

<company> is also having this <process> thing which is kind of forcing us to fix technical debts, or basically that we are getting penalty points if we are not fixing technical debts in kind of reasonable time. So that is also kind of helping us prioritizing those technical debts. (6)

This explains that there are factors inside the company that contribute to faster repayment of technical debt. From what the respondents have explained are most of them on level 2 as many of them continuously repay technical debt as part of the daily work. The **team size** can also have an impact here as three respondents mentioned this. Respondent 2 said the following:

It boils down to how every team feels to do that. If you have a larger team then you have a higher chance to repay them faster. If you have a small team you have a chance of getting into those penalty issues and that's what happened in my case right now. (2)

Respondent 2 points out that having a larger team puts you in a better position to repay technical debt and can therefore avoid getting too many penalty points. A small number of respondents said that they do not repay technical debt continuously. One of the respondents that said this pointed out that they only choose to repay technical debt when it becomes a problem. As long as it does not disturb, it will not be fixed. On the other hand, respondent 22 explained the importance of repaying technical debt before continuing adding features etc.:

... you cannot put paint on, let's say, broken walls. If you have a broken wall then fix it first. You know that's broken, your paint won't cover the entire bad internal structure. (22)

This example points out that it is hard to create a nice system that has a lot of problems. Fixing and repaying technical debt and other issues must be done before continuing adding functionalities etc..

None of the respondents mentioned that they currently use percentages when repaying security debt but there is a general understanding that security issues will be fixed first and this includes security debt. Respondent 11 did on the other hand talk about dividing the work into percentages in the future so that you can keep track of how much time you spend on different tasks:

... of course you need some way to measure how much time you're using for each or just try and balance them in a certain way which is much harder of course if you're not measuring. But for the balance I think you could probably go with 50% on paying off debt also including fixing old bugs and 50% of developing new features including fixing bugs in those features immediately. (11)

Respondent 11 is the only respondent that mentioned a specific percentage for the security debt process. The percentage that this respondent mentioned is not only for security debt but for debt in general. In the technical debt process, the recommendation is using 20% of the team's capacity for the repayment of technical debt. The percentage mentioned by respondent 11 is then 30% higher than what is recommended in the technical debt process ($50\% - 20\% = 30\%$). The respondent explained that it is not a question of frequency of the repayment of security debt but rather how much time is spent on it.

A summary of the key points:

- 4 respondents mentioned a specific percentage for technical debt repayment but most of the respondents repay technical debt as part of their daily work.
- Penalty points and team sized can have an affect on the technical debt repayment.
- Only one respondent spoke of a percentage for the security debt process (50% for the repayment of debt including bug fixes).

4.3.9 Additional observations - security self-assessment

In addition to the majority of the respondents thinking that the technical debt process can be used for security debt, there are also a few people that think that the **security self-assessment** in the company can be used (mostly for the identification of security debt). This assessment looks at how the teams work with security. Respondent 7 explains that

we are also doing this security self-assessment which basically is a lot of questions that we have to answer, and in that we have to go over the application to see if there are any known [security] issues, if there are other stuff. (7)

The security self-assessment contains a list of things the teams have to go through, e.g. listing the attack surfaces, password storage, cryptography/ hashing algorithms, security logging, quality assurance and testing etc.. The questions asked during the self-assessment aims to find out if the teams have/do the needed security. Doing this security assessment highlights security issues in a way that the technical debt process do not. Respondent 21 points out the following:

Technical debt assessment is mostly focusing on the general process but since security is prioritized normally higher than other technical debts then it's security self-assessment. (21)

When security debt is found during the security assessment it is ranked differently when it comes to the penalty points. Respondent 2 said that the security debt then have a higher percentage, meaning that they get more penalty points on these issues depending on how long they have been left unassigned or unattended. This is, as explained above, a factor that influence the repayment. Both processes (the technical debt process and the security assessment) agree that **it is important that security debt is prioritized**. A few respondents did not mention a specific process and one of these respondents said that they are not in a position to make a security debt process.

4.4 Security debt prioritization (RQ2.1)

There is a need for prioritization in order to find out what work will be done in the next period of time. Respondent 11 points out that there is a trade-off as a result of limited amount of development resources. The prioritization of the repayment of security debt is, according to almost all of the respondents, done by setting priority and/or severity (this points to levels 2 and 3 in the analysis section in the technical debt process). The use of severity and priority was explained in subsection 4.3.4. Respondent 21 points out the following regarding **evaluation**:

at least they [security debt] always get on the top of evaluating. They might not always be on the top when it comes to the fixing but when it comes to evaluating and prioritizing, they are always on the top. (21)

After the evaluation of the security debt, it is **prioritized**. The security debt's level of importance ranges from "fix it now" to "fix it later" and this is something that can be seen with the prioritization it is given. Even though almost all respondents mentioned using priority and/or severity to prioritize the security debt, mentioned 12 respondents the importance of prioritizing security issues and 6 respondents said that security issues could disrupt the sprint. Respondent 22 and 6 said the following:

I'm trying to do this priority: it's security, then it's technical debt, then bug, then feature. ...of course I cannot do only NFR and the security, I have to make features because I have to sell things but I'd rather postpone a feature until I have a sane foundation then deliver it based on, let's say, compromised framework or something. (22)

Security issues are always prioritized at the top and basically security related issues are the ones that are able to also ruin our sprint plan, so security issues might appear in the middle of the sprint, and we need to get them prioritized, which means that something else is down prioritized (6)

The respondents quoted above explain that security fixes goes before everything else. Respondent 25 agree with this and explain that even with less likelihood and impact it should be fixed:

... you shouldn't have much security debt I will say. As I said it's high priority even if it's less likelihood that it can happen and it can destroy, or it can affect that someone is stealing your data then it's very important. (25)

As shown above there is a general agreement that security issues are fixed first, where these issues even have to ability disrupt sprints. Even security debt with less likelihood is something that should be prioritized according to respondent 25.

Cases/scenarios where security debt repayment is not prioritized
I asked the respondents whether or not they had any cases/scenarios where the repayment of security debt is not prioritized because many of the respondents said that security related issues was something that they choose to fix fast. 11 respondents had examples where this was the case. Respondent 1 and 9 said the following:

For us, production issues have higher priority. So if someone says that you have a bug in production so that we are not able to update files then we fix that. And then when it comes to features and security debt it is an even distribution. (1)

Scenarios where security debt issues are not fixed:
***1)** If the severity is less, or if it's not high and it's a bigger job.*
***2)** If there is a security solution that requires changes to the cooperating system, and the third party is not willing to change (e.g. the bank only supports this type of SSH key or this type of connection).* (9)

In these cases security debt is something that did not get on the top of the prioritization because of more pressing issues or that the available solution is very hard to do. Respondent 5 is one of the respondents that mentioned that it is important to keep the customers and their needs in mind. The respondent explained that in some instances a super important security issue must be put on hold because the customer wants a feature. The reason for this is that if there are no customers because of a lack of features then there is no point in fixing the security issues. This can also be viewed from the opposite perspective; there is no point in having a service that has many nice features if it is not secure.

Backlog prioritization
The primary responsible roles for the planning and the prioritization of technical debt repayment is the product owner/service owner (PO/SO) as they communicate to the development team and the stakeholders. In order for the backlog to be prioritized the PO/SO needs to know what is important from different perspectives. 12 respondent said that they have good **communication** with the PO/SO and that they discuss the backlog prioritization with them. Respondent 21, 16, and 18 said:

...it's also good for us because when you have this routine [communication], when you do something on a regular basis it also helps you to have a better overview and of course talking to this service owner is also better communication because the service owner can tell us what he sees from his perspective because that's also important, it's a reflection of stakeholders, etc. and that helps also him to understand how we see the things, it's also important (21)

...we have a meeting every two weeks with me [architect] and the service owner where we prioritize the backlog and based on that prioritization we have another meeting with the entire team and discuss the tasks that were prioritized. (16)

usually it it's like there is a conversation in slack before creating a task and during that we prioritize it usually together. (18)

As explained above, there is a continuous communication between, not only the architects and the PO/SO, but also with the rest of the team. In addition, respondent 18 explains that they also discuss the security debt items in the team before they are created. This way everyone can give input and share their knowledge.

Communication between team-members, the architect and the PO/SO is not the only thing that contribute to the backlog prioritization. 12 of the respondents said that the labelling of the security debt items also helps with the prioritization. The most mentioned label was the "security" label. Respondent 14 and 22 explained that the security label is the label that prevails and shows importance as security issues typically takes precedence.

To summarize, the governing factors that helps the PO/SO prioritize the backlog is:

- **Communication** between team-members, the architects, the PO/SO, and the stakeholders
- The **labels** that are added to the Jira tickets
- The **severity/priority** score added to the Jira tickets

4.5 Security knowledge (RQ2.2)

I asked the respondents how they view the importance of having security knowledge in order to handle security debt. There is a general consensus that security knowledge is important and is described by the respondents as a range from very important to something that is case dependent. A number of the sections in the technical debt/security debt process have been mentioned in connection to security knowledge and can be viewed in table 4.4.

Process section	**Number of respondents**
Prevention	11
Identification	24
Evaluation	2
Prioritization	5
Repayment	22

Table 4.4: Number of respondents who view security knowledge as important per mentioned process activity.

Prevention

The first section of the technical debt/security debt process is having the ability to prevent them. 11 respondents mentioned the importance of having security knowledge in order to be able to prevent security debt. Respondents 2 and 14 said the following:

Of course, because first of all you avoid creating that issue in the first place if you know how to avoid it and secondly you understand why it is needed when it is reported to you. And if you don't know it then you discuss it... (2)

... also designing to reduce risks. One example is in our service: do we need to work with social security numbers? Well we have to, you know under GDPR, we have to work with them in the sense that they're part of a data set that passes through our system... We do need some way of joining information about individuals from different datasets. It turns out that social security number is the one common factor. ...what we did instead [of storing the social security numbers] was implement strong hashing using encryption and so that what we store is an encrypted, salted version so that if for some reason that data were to leak at least we wouldn't leak that part. That reduces the impact of a security incident. (14)

The example described by respondent 14 pointed out that thinking through possible attacks helps them design and implement the system to reduce risks. From earlier it was stated that 16 of the respondents said that they are part of the bug bounty program and that this is a good source for being notified of security issues, including security debt. Respondent 9 emphasizes that having good security knowledge in order to prevent the accumulation of security debt is reflected in the fact that they had very few reported security issues during the bug bounty and that those weren't even critical.

Identification

The next section in the development process where the respondents found it important with security knowledge is identification. 24 respondents stated that having good security knowledge is important when it comes to being able to identify security debt. Respondent 8 said the following:

You definitely needed for identifying. I mean if you don't know what you're looking for... I mean something is missing right, or it can be done in a much more elegant way that is how it's supposed to be done according to the system that you are using then you need to know about these things. If you don't then you don't see a problem, right. (8)

Respondent 21 pointed out that it is not possible to look for something you are do not know of. Earlier it was mentioned that some respondents use code reviews in order to identify security debt. Respondent 13 and 14 are two of them and explained that the team always goes through a code review and that having good security knowledge is important in order to catch possible security issues.

Evaluation

After the identification of security debt it is important to do correct evaluation of the security debt so that it gets the proper attention. Respondent 18 is one of the respondents that mentioned this and explains the following about evaluation:

Without security knowledge you can't evaluate correctly. ... First of all, evaluation is affected. And yeah, security knowledge helps you fix it quicker, better understanding overall for you to fix. (18)

This respondent explains that not only is the evaluation affected by security knowledge but also the ability to understand that something needs to be repaid. This understanding affects how the prioritization is done.

Prioritization

5 respondents spoke about security knowledge as a means for being able to do correct prioritizations. Respondents 3 and 21 pointed out that:

Knowledge is a tool-set to being able to recognize and prioritize the issues. Without knowledge, the prioritization becomes guesstimations. (3)

The better knowledge the better understanding the consequences, so the critical issues will be fixed first while the minor can be postponed in favour of other tasks. (21)

Both respondents highlight the need for security knowledge for doing correct prioritizations because the most pressing issues should be fixed first. When the issues have been prioritized it is time for repayment.

Repayment
22 respondents talked about security knowledge for the repayment of security debt. This group of respondents can be divided in two; the ones that view security knowledge as a must and the ones that think that it is dependent on the issue at hand. Respondent 7 is from the first group and respondent 8 is from the second group:

[Importance of having security knowledge] much because if they have an understanding of what the problem is, not only makes it fixing it easier, but it also makes them more aware of the problem and if they're aware of the problem it's more motivation to get fixed. (7)

...of course you need to know something when you're solving them but you can always be told what to do. I mean if I follow the steps 1,2,3 and make sure of this. So it depends on the case at hand and what you're actually doing. Are you implementing? I can write you what to do- follow these steps, do this. (8)

Both respondents view security knowledge as important but respondent 7 think of it as more important than respondent 8. Respondent 7 mentions the motivational factor of wanting to repay the security debt because they then understand the risk.

The sharing of security knowledge
In addition to having security knowledge for the prevention, identification, evaluation, prioritization, and the repayment of security debt, it is also mentioned by 11 respondents that sharing this security knowledge is important. Respondent 15 points out:

...it's very helpful [sharing security knowledge] because you can start with your idea and others see it from a different point of view and all of the sudden you see a different light. (15)

Having this culture of sharing knowledge can help non security people to better understand how to create secure systems. Three respondents specifically mentioned that having a **security mindset** helps the production of secure systems. Respondent 4 explains the need for this security mindset:

> *...we often challenge security when we design things. Let's call it a security mindset, the developers kind of need to have it nowadays because you see we have the pandemic and we work from home, and security is more and more important as things are getting more and more online and automated. So you need as a developer to have this mindset, and not just the developers but QA's as well, every role I would say.* (4)

Several of the respondents have mentioned the importance of security knowledge for more than one of the sections in the security debt process. Respondent 2 for example is one of the respondents that point out the importance of having security knowledge for all the sections mentioned above. All the respondents that talked about security knowledge for the different section said that it is important. The only exception is the repayment section, here there are divided opinions. Some say that it is very important as you cannot repay something you do not understand while others point out that it is possible to tell someone what to do. This is related to the sharing of knowledge. If someone explains how to solve a problem then that knowledge has been imparted to the person who previously did not know how to solve the problem at hand. It can be argued that lacking the security knowledge for handling security debt can be a cause for the accumulation of it. As pointed out by respondents 2 and 8, it is not possible to prevent something if they do not know what it is and they cannot identify something when they do not know what they are looking for. It is tricky knowing if the teams have the needed security knowledge for handling security debt as there is no final answer that shows what security debt is being accumulated.

It is mentioned by respondent 7 that if a person have the right security knowledge then they are able to understand the risk of the solutions that are being implemented. If these implementations are less then optimal then they can notice this and prevent them. The understanding of the risk is a motivational factor for repayment. Respondent 14 agree with respondent 7, where respondent 14 point out that having security knowledge helps reduce risks because they have the ability to be aware of them because of this knowledge.

The importance of security knowledge can then, as a result of the respondents that mentioned the different activities, be simply put:

- **Prevention**: security knowledge is important as it is difficult to prevent something the team is not aware of. Having security knowledge also helps the team design and implement the system to reduce risks.
- **Identification**: security knowledge is important as it is explained that it is hard to identify something the team does not have any knowledge of.
- **Evaluation**: security knowledge is important because the team is then able to do correct evaluation and have the needed knowledge to understand that an issue needs fixing.
- **Prioritization**: security knowledge is important because without it the evaluation becomes guesstimations and a lack of security knowledge will not provide the needed understanding of the consequences.
- **Repayment**: the need for security knowledge is case dependent. In certain situations it is enough to explain how an issue can be fixed while in other instances having the needed security knowledge can be beneficial.

4.6 Relation between security debt and technical debt (RQ3)

This section is divided in three; 1) relation between technical debt, security debt, and security vulnerabilities, 2) security debt and architectural technical debt, and 3) the technical debt process for the different kinds of technical debt.

4.6.1 Technical debt, security debt, and security vulnerabilities

During the interviews I got information on the relation between security debt and technical debt.

Technical debt description
In the beginning of the interview I asked the respondents how they thought of technical debt and if they had any examples. I did this so that I could get

insight into how the respondents view technical debt and to make sure that they were familiar with the term before continuing the interview. This type of information is valuable as it can give an indication to how they think of security debt, if there are any common areas. Their answers were divided into four main groups: deliberate, not deliberate, due to change in technology, and its effects on the maintainability of the system. These groups were created as a result of either specific wording or how respondents did the overall explanations (e.g. respondent 12 specifically mentioned that technical debt cause maintainability issues). The number of respondents in each group can be seen in table 4.5.

Group description	**Number of respondents**
Deliberate technical debt	22
Not deliberate technical debt	15
Technical debt effect on maintainability	15
Technical debt due to change in technology	10

Table 4.5: Groups that explain how the respondents think of technical debt and the number of respondents per group.

During the coding of the interviews I found that most of the respondents thought of technical debt as being something deliberate, meaning that it was an active choice they did during development. Not deliberate technical debt and maintainability issues due to technical debt had the same number of agreeing respondents. There was also a number of respondents that said that technical debt absolutely could be deliberate but that was not something they did anymore. It was something they did while starting out and wanted to get the service up and running. The group with the least amount of respondents is that technical debt is due to change in technology. Respondents 2, 23, 16, and 5 explained the following about the four groups presented in table 4.5, deliberate technical debt, not deliberate technical debt, technical debt effect on maintainability, and technical debt due to change in technology:

I know I need to do something to improve what I did here because what I did here was done under certain circumstances, either lack of time, pressure, I have to deliver it, had to be done fast, or simply I realize I did it wrong, so I feel an obligation of fixing this at some point in time. (2)

Nowadays it's the first one [not deliberate technical debt], that we don't know *and we later discover that- oh there is actually something that we need to handle.* (23)

Any short cuts that you do in the code in order to release it [the system] faster might be considered technical debt if that shortcut, I don't know, introduces some hax or the code base will not be easy to maintain from that moment on. (16)

[technical debt] It's not something you create, it's something that happens because assumptions change, environment change, everything changes. (5)

Security debt and technical debt relationship

The groups in the table show that there is a relation between technical debt and security debt as three of the same groups were used to categorize security debt (as explained in section 4.1). It is said by several respondents that technical debt and security debt indeed are connected, where two of them are respondents 5 and 1. Respondent 5 pointed out that security debt is a part of the overall technical debt and respondent 1 explained the relation between them as the following:

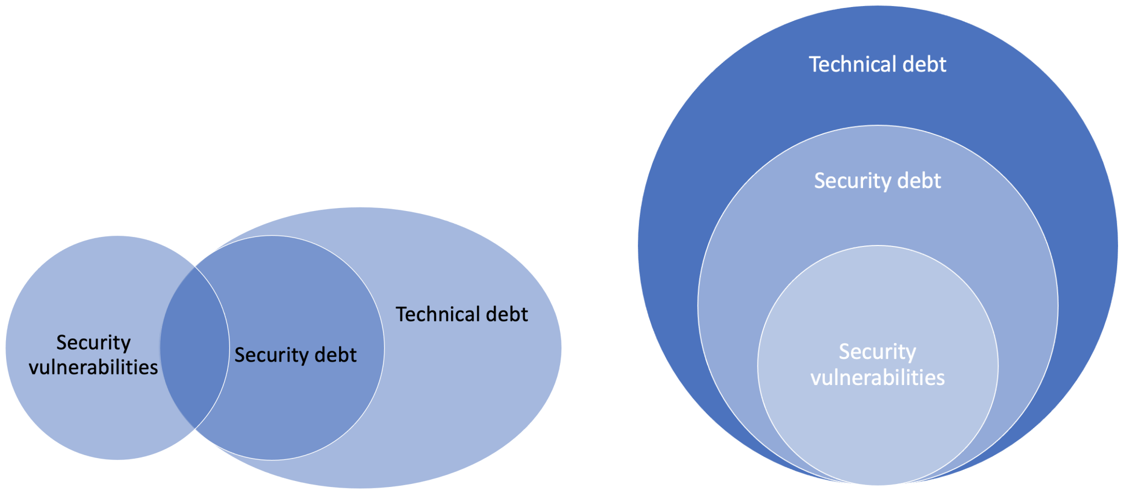
they [security debt] are connected to technical debt indeed. They are a debt in a way. It's still technical, just that they are also security. (1)

Group 3 from table 4.1 describe security debt as the need to do updates. I asked the respondents if they had any examples of technical debt and 11 respondents considered technical debt as having to do various updates, among other things. There were common respondents between group 3 from table 4.1 and the technical debt examples. The ones that mentioned the TLS example as a security debt also spoke of the TLS example as a technical debt. As the same example is used for describing both security debt and technical debt it can indicate that some respondents view some debts as both security debt and technical debt. The update can be done to reach a higher security goal as there exist a better solution.

Other respondents view the occurrence of technical debt and security debt as different. Respondent 6 said that technical debt can be due to shortcuts while they would never take shortcuts when it comes to security. Respondent 2 explain that even though security debt and technical debt exist in the same pool of technical debt, security debt has a higher priority.

Relation between security debt, security vulnerabilities, and technical debt

The relation between security debt and security vulnerabilities was discussed in section 4.2. As there are several respondents that explain that there is a relation between technical debt and security debt can the figures 4.1 be expanded. Figure 4.3 shows the expanded figures that present the relation between security debt, security vulnerabilities, and technical debt.



(a) (b)

Figure 4.3: Two visible representations of the relation between security debt, security vulnerabilities, and technical debt. Figure (a): technical debt completely surround security debt and have the same shared area as security debt and security vulnerabilities. Figure (b): technical debt completely surrounds security debt and security debt completely surrounds security vulnerabilities.

The 4.3a figure shows that technical debt overlap with both security debt and security vulnerabilities and is shown to contain all of the security debt. Figure 4.3b also show that technical debt contain all security debt. Technical debt is, in this figure, the superset. Security debt is a subset of technical debt and security vulnerabilities are a subset of security debt.

4.6.2 Security debt and architectural technical debt

I asked the respondents whether or not they had experienced having architectural issues that impacted the security. There was a fairly even distribution of yes and no and one said that they did not remember. Respondent 21 explained that *we are moving all our secrets from one location we had in our deployment pipeline, it was stored in the Octopus libraries, and now we are moving to the Azure key vault which gives better security, better protection...* (21)

This is an example that was specific to that team while several respondents, including respondent 6, mentioned the TLS example from earlier, where an older protocol version can be used against them. They explained this as an architectural decision that had a security impact. The effect that architecture and security have on each other is described by the respondents from **"somewhat"** to **"a lot"**. Respondent 11 said the following:

That's [architecture and security effect on each other] from somewhat to very much, I mean you got the whole scale there. It depends on what you want to do. We don't expose much data that isn't the customers' own data so we don't need to take as much care as you would if you're managing others' data, perhaps even on offline devices which could affect your architecture severely. But for sure they both affect one another. If you have a broken architecture, that could be used in an attack. If you want to be safe against attacks, you need the architecture to reflect a secure design. So they both affect each other. (11)

It is explained that the relation between architecture and security is dependent on the needs of the system but that they affect each other because the architecture needs to reflect a secure design.

Architectural technical debt and security debt

I added the debt perspective to the question and asked the respondents how they thought architectural technical debt and security debt effected each other. They had a few different descriptions of the relation between them. Most of the respondents (9 respondents) that had a clear answer described the relation between architectural technical debt and security debt in the same way: architectural technical debt could lead to security debt. This is explained by respondent 14 and 15:

So if for some reason you chose the wrong architecture it can probably land you in all kinds of you know security issues, particularly related to data flow for instance. (14)

Yes because you can have an early architectural decision that will hit you up *in time. ... it's just you started that way, you haven't thought enough. I think it's a minus of architecture that becomes a security debt.* (15)

These explanations point out that a wrong or hasty architecture can result in security issues. Respondent 17 is another respondent that think that architectural technical debt can lead to security debt and pointed out that the attack surface will increase when the architecture gets older and is not properly maintained. Respondent 1 mention changes in both the attack surface and the data flow but does not necessarily see a causality between architectural technical debt and security debt. The following was explained:

An architectural technical debt does not necessarily imply a security debt. But the changes that are done to the system once that architectural debt is worked on can have a security impact. Architectural changes coming from technical debt usually trigger a change in the data flow diagram and the attack surface of the system changes. Architectural changes can of course be triggered by a security debt that requires a larger rewrite of a part of the system. So I don't think that all technical debt is also a security debt or the other way around. You can have one without the other. But the changes done while working at resolving the debt can lead to other security vulnerabilities. This is the reason why, while doing architectural changes, teams need to threat model these changes to make sure they don't introduce security debt. (1)

Respondent 1 points out that there is a relation between security debt and technical debt but that one can exist without the other. In addition to this is it pointed out that changes in the architecture as a result of architectural technical debt can trigger security issues and the repayment of larger security debts can trigger architectural changes. Respondent 21 had an example that showed this. It was explained that they needed to change their approach to storing their secrets. This was found as a security debt because they needed to improve the security by using a better solution and this improvement resulted in changes in the architecture.

In addition to the explanation above there is a disagreement between respondent 22 and 24. Respondent 24 pointed out that if they do not have a secure system then they do not have a good architecture because "*security is one of the quality criteria of architecture*". On the other hand respondent 22 said that it is possible to have a secure system with bad architecture and a non-secure system with nice architecture. While they think differently on this, both agree that in cases of trade-offs security should prevail. Respondent 22 mention that architectural technical debt and security debt go hand in hand but that they are "*not caused by the same causes*".

A set of 7 respondents that was asked about the effect just like the other respondents did not provide an answer in the context of debt but rather just in the context of security and architecture.

From what has been described above, the main take-away points are:

- The effect that architecture and security have on each other is described from "somewhat" to "a lot".
- There is evidence that there is a relation between architectural technical debt and security debt but there is not necessarily a causality between them.
- Architectural technical debt can lead to security debt.
- Changes in the architecture due to architectural technical debt can cause security issues and the repayment of security debt can trigger architectural changes.
- 7 respondents did not talk about architectural technical debt and security debt together.

4.6.3 Technical debt processes for different types of technical debt

I asked the respondents whether or not they follow the same technical debt process for all the different kinds of technical debt. The response was that 15 respondents agreed that the same technical debt process could be used while 11 thought that there should be a difference between the handling of for example code debt and architectural technical debt. The reason for this was that they have different complexities and therefore should be handled a bit differently. Respondent 3 said the following:

I would say that in code it is different because here we have these tools. But architecture is like I would say finding a needle in a haystack. (3)

Group 5 from table 4.1 explained security debt as "security debt are security cases that do not have straight forward fixes". Just like with architectural technical debt there is a complexity factor that should be taken into consideration (8 respondents agree that complexity should be taken into consideration when it comes to the management of the debt). Security debt is also mentioned to having higher priority. It can be argued that if the technical debt process can factor in the complexity of architectural technical debt then it should also be able to factor in the complexity of security debt (as explained by group 5) and the need for a higher priority.

Chapter 5

Discussion

In this chapter I will be discussing the findings presented in chapter 4 together with the background and related work from chapter 2. I will be following the order of the research questions found in section 1.2

5.1 RQ1 How is security debt defined?

Security debt is a term that has been attracting an increasing amount of attention by both practitioners and researchers. (Martinez et al., 2021). Some of the publications that discuss security debt are Martinez et al., 2021; Rindell et al., 2019; Rindell and Holvitie, 2019; Silva et al., 2016. This case study provides new input on the area of security debt. Being able to avoid security debt or being able to use it strategically is dependent on security debt being properly defined and people having a common understanding of what it is.

The aim of the first research question is to discuss a definition of security debt. Based on the inputs from the respondents, this study proposed the following definition in the results chapter (section 4.1.1):

Security debt is a set of design or implementation solutions that hinder or has the potential to hinder the achievement of a system's optimal/desired/required security goal.

Four publications that discussed security debt definitions are Martinez et al., 2021; Rindell et al., 2019; Rindell and Holvitie, 2019 and Silva et al., 2016.

For the security debt definition,

- Rindell et al., 2019 focused on the connection between technical debt and security risks,
- Silva et al., 2016 described solutions that compromise the security, and
- Rindell and Holvitie, 2019 pointed out a two-fold for defining security debt: (1) technical debt found through security verification or validation methods, and (2) technical debt in a software component that is critical to the security.

Both Rindell et al., 2019; Rindell and Holvitie, 2019 described security debt to be in direct relation to technical debt while Silva et al., 2016 describes a solution that is not optimal in regards to the system's needed security. This is in line with groups 6 and 7 from table 4.1:

- Group 6: Security debt is related to technical debt
- Group 7: Security debt is described as security levels

The three studies presented above were part of the data analysis entries that Martinez et al., 2021 used when defining security debt. Their definition is as follows: "*security debt is incurred when limited approaches or solutions are applied (intentionally or unintentionally) to reach the needed security levels for the system in operation*" (Martinez et al., 2021, p. 2).

The goal of this study was to achieve an independent security debt definition. However, the thesis definition ended up not being hugely different from the definition produced by Martinez et al., 2021. Both definitions speak of wanting to reach a specific security level/goal for the system. Not having the needed security due to security debt makes the systems more susceptible to malicious attacks (Martinez et al., 2021), meaning that security debt have a negative impact on the system. It is therefore important to identify the environment in which the system will run so that the development team know which security controls are needed (Maymi & Harris, 2019, p. 1084) to reach the security goal. A key difference between the definitions is that Martinez et al., 2021 mentioned that limited approaches for reaching the security level is considered to be security debt. The definition presented in this thesis on the other hand does not specifically mention this, as only one respondent spoke of limited approaches (e.g. tooling and intrusion detection systems) as security debt, it did not have a central part in the definition. Another difference is that the thesis definition points out that security debt does not necessarily hinder the achievement of the system's security goal but rather has the potential to. This means that as soon as the postponed internal security issues are exposed to environments where they can be misused, they become security debt (the security issues did not pose any danger when they were internal). The system is then more susceptible to malicious attacks because of the security issues that have now become security debt.

The main take away points between the thesis security debt definition and the latest literature definition by Martinez et al., 2021 is:

- Martinez et al., 2021 mentions that security debt is incurred when having limited approaches to reach the security level.
- The thesis definition describe that security debt not only hinder but also has the potential to hinder the achievement of the wanted security goal.
- Both definitions describe security debt to be detrimental as it makes the systems more susceptible to malicious attacks.

5.2 RQ1.1 What is the difference between security debt and security vulnerabilities?

The focus of the sub-research question, RQ1.1, is finding the difference between security debt and security vulnerabilities. The groups of explanations presented in the results chapter is shown in table 4.2.

A vulnerability is explained as "*a weakness in a system that allows a threat source to compromise its security*" (Maymi & Harris, 2019, p. 6). This definition of vulnerability focus directly on security compromise. Thus, it is fair to argue that the definition presented by Maymi and Harris, 2019 also includes security vulnerability. What is then the relation to the proposed definition of security debt? This definition explains security debt as **a set of design or implementation solutions that hinder or has the potential to hinder the achievement of a system's optimal/desired/required security goal**.

The definition of vulnerability (security vulnerability) and security debt presented above indicate a close link between the two as shown in figure 4.1. This is supported by the key findings presented in the results. The security vulnerabilities with existing solutions can be considered as security debt because the development team can fix the weakness in the system and reach for the system's security goal. It follows from this discussion that if a security vulnerability does not have an existing solution then it is not considered as security debt by the proposed definition.

More specifically, a security vulnerability is considered as a security debt only after it is merged and materialized in the code (see results 4.3). This can be compared to **defect debt**. Defect debt is "*the trade-off between the short-term benefit of postponing bug fixing activities and long-term consequence of delaying those activities*" (Akbarinasaji et al., 2016, p. 1). This means that the identified failures, defects, and bugs that are not fixes in the same release as they are found are defect debt (Akbarinasaji et al., 2016). Not fixing (postponing) the security vulnerabilities might be a choice (which results in deliberate security debt) while in other cases the team might not know about the security vulnerabilities (in that case it is not deliberate security debt). Respondent 7 explained that depending on how fast the security debt is repaid it is either a short-term debt or a long-term debt (depending on the prioritization). Even though the security vulnerabilities are considered to be security debt during a period of time after they have been materialized in the code, they are still considered to be security vulnerabilities as well as security debt. Fixing one of them will automatically fix the other as it is the same issue.

The proposed relations between security debt and security vulnerabilities

There are two proposed relations between security debt and security vulnerabilities as shown by figure 4.1. Three sides to figure 4.1a was presented in the results chapter. In the following, a fourth side is included:

1) Security vulnerabilities with existing solutions that are not fixed in the same release as they were found (postponed) is security debt (this can be compared to defect debt).
2) Security vulnerabilities that do not have existing solutions are not security debt.
3) Other security issues that are not security vulnerabilities can be security debt.
4) Security vulnerabilities with existing solutions that are fixed in the same release as they were found does not become security debt because they

are fixed before they are materialized in the code.

Both figures 4.1a and 4.1b show that **security debt can be something other than only security vulnerabilities** but figure 4.1b show that **all security vulnerabilities are also security debt**.

The first proposed relation between security debt and security vulnerabilities (figure 4.1a) corresponds the most with the thesis definition of security debt. This is due to security debt being explained as issues that have better existing solutions than the ones currently implemented, meaning that if an issue, e.g. a security vulnerability, does not have a solution then it cannot be a security debt. This is shown by 4.1a; security vulnerabilities can exist without being security debt. These security vulnerabilities then have no solutions and therefore cannot be improved, which is in line with the security debt definition proposed in this study.

It seems from the discussion that figure 4.1a shows the most descriptive relation between security debt and security vulnerabilities due to the security debt definition. This is supported by the fact that figure 4.1b does not take into account the security vulnerabilities that do not have existing solutions (and therefore cannot be security debt). Having several ways of understanding the relation between security debt and security vulnerabilities can negatively impact organizations. It is imperative that everyone understands concepts in the same way to be able to work towards the common goal of security.

5.3 RQ2 How is security debt managed?

The aim of the second research question is to look into how security debt should be managed. I asked the respondents how they would manage security debt and the answer with the highest number of respondents was that their already existing technical debt process would work for security debt. The general consensus was that the difference between them is that **security debt have a higher priority than technical debt**.

During the management of technical debt, a set of activities is performed. The document I received from the company explaining the technical debt process referred to prevention, identification, documentation, etc. as sections. In this discussion I will call them activities (as it is also explained in the literature). The 8 activities (sections) in the company's technical debt process corresponds with the activities found by Li et al., 2015. The additional three activities time-to-market analysis, scenario analysis, and visualization from Rios et al., 2018 are not part of this process. Li et al., 2015 spoke of approaches that are performed during the activities while Rios et al., 2018 called the guiding of the activities' execution strategies. In this thesis I am referring to both of them as approaches.

As **the process for technical debt can be used for security debt**, the same activities apply. An overview of the 8 security debt process activities presented in the results chapter can be seen in table 4.3. There are three additional approaches that were specifically mentioned in connection to the security debt process. These are

- threat modelling,
- the use of a bug bounty program, and
- security testing performed by the company's security team.

I will go through the activities and their maturity level 3 approaches in the order they were presented in the results.

5.3.1 Prevention

Technical debt "*prevention aims to prevent potential TD from being incurred*" (Li et al., 2015, p. 204). During the prevention of technical debt, the company approaches are following **coding standards** and performing **code reviews** for non-trivial changes. Some of the approaches for preventing technical debt mentioned in three earlier publications are:

- human factors analysis (Li et al., 2015)
- adoption of good practices (Freire et al., 2020; Pérez et al., 2021)
- code evaluations/standardization, and training (code review/refactoring) (Pérez et al., 2021)

The findings in this thesis agree that the approaches code standardization and code reviews presented by Pérez et al., 2021 can be used for preventing technical debt. Tom et al., 2013 explains that having code reviews makes the developers less likely to make decisions that increases the accumulation of technical debt.

These same technical debt approaches are said to be used during the security debt process, including threat modelling. **Code reviews** are described to be essential for making sure that the software is of good quality (Maymi & Harris, 2019, p. 1084). As both technical debt (Kruchten et al., 2012) and security (ISO/IEC, 2011) is related to software quality, this can be helpful when preventing security debt. Following coding standards can not only be used for preventing technical debt but adopting **secure coding standards** (Maymi & Harris, 2019, p. 1122) can help prevent security debt from accumulating as well. Validating input, sanitizing data, keeping the code simple, etc. are practices that can be used as part of the secure coding standard (Maymi & Harris, 2019, p. 1122). **Threat modelling** is the last mentioned approach. Threat modelling is "*the process of describing feasible adverse effects on our assets caused by threat sources*" (Maymi & Harris, 2019, p. 97) and can help prevent security debt. Having identified and described the effects the threat sources can have on the assets can help understand how to develop effective defenses (Maymi & Harris, 2019, p. 97) and prevent the accumulation of security debt. Respondent 1 said that modelling the threats when doing changes in the architecture is needed in order to make sure that security debt is not introduced.

5.3.2 Identification

Technical debt "*identification detects TD caused by intentional or unintentional technical decisions in a software system through specific techniques, such as static code analysis*" (Li et al., 2015, p. 204). The company's technical debt process have three approaches that can be used for identifying technical debt. These approaches include

- identifying technical debt during implementation and code reviews (explained above),
- by the means of static analysis tools, and
- as part of operational processes (e.g. incident reviews and problem management).

Two mentioned approaches for identifying technical debt from the literature are code analysis and check list (Li et al., 2015). The **code analysis** approach matches both the first and the second approach from the company's identification activity. Source code analysis is done in order "*to identify violations of coding rules, lack of tests; calculate software metrics based on source code to identify design or architecture issues*" (Li et al., 2015, p. 206).

A number of tools for identifying technical debt have been found, such as DebtFlag, FindBugs, Sonar TD plugin, CodeVizard, and SonarQube (Li et al., 2015). The three most mentioned tools utilized in the researched company is **Coverity**[1], **SonarQube**[2], and **Snyk**[3]. A few respondents also spoke of using **Linters**[4] for flagging different bugs, errors, etc..

These technical debt approaches are also said to be used for managing security debt. As explained above, performing **code reviews** is a means for identifying and preventing security debt. Security debt can also be **identified during implementation** as the developers are then looking at code. Two respondents mentioned that this is quite difficult. This is something I will explain further in section 5.5. The main **tools** mentioned by the respondents (SonarQube, Coverity, and Snyk) are also able to detect security issues in the code. Having tools that can identify vulnerabilities is valuable as it has also been explained that security vulnerabilities can be security debt. Part of the **operational processes** are incident reviews and problem management. After an attack has been mitigated, the remediation phase tries to makes sure that that kind of attack is never successfully carried out again. The incident is reviewed in the hopes of learning how to avoid those kinds of incidents and to find out how to do better the next time (Maymi & Harris, 2019, pp. 1008–1009). This can be a way of identifying security debt if similar security issues have been reported in earlier incident reviews. In order for these incident reviews to be beneficial is it important that the people doing the reviews have the needed knowledge to spot the important issues. This can be compared to the technical debt identification approach **check list**- "*check against a list of predefined scenarios where TD is incurred*" (Li et al., 2015, p. 206).

[1]https://www.synopsys.com
[2]https://www.sonarqube.org
[3]https://snyk.io
[4]Linters "*check the occurrence of textual or syntactical information on pre-defined code rules*" (Schreiber et al., 2021)

The second to last approach for identifying security debt is the use of a **bug bounty program**. Bug bounty programs allows ethical hackers to hack an organization's system so that issues the organization's internal security people was not able to find can be identified (Laszka et al., 2018). Some of the issues that are identified during the bug bounties can be security debt. The last approach is **security testing** by the company's security team. Three of the four respondents that mentioned security testing specifically mentioned **penetration testing**. During this kind of testing, attacks are simulated in order to try to get around a systems security controls and to measure the resistance level of the organization (Maymi & Harris, 2019, pp. 873–875). Another kind of testing that can be performed is **vulnerability testing**. This kind of testing only identifies vulnerabilities that potentially can be exploited, while during penetration testing the vulnerabilities are exploited in order to show that it is possible to gain access to the system (Maymi & Harris, 2019, p. 878).

5.3.3 Documentation

Technical debt "*representation/documentation provides a way to represent and codify TD in a uniform manner addressing the concerns of particular stakeholders*" (Li et al., 2015, p. 205). Only one approach is listed for this company activity, namely that the technical debt is to be documented just like everything else: in the same **backlog**. When technical debt is documented, a number of fields can be filled out, e.g. an ID, where the technical debt is located, the person responsible for the repayment, what type of technical debt it is, a brief description of the technical debt, etc. (Li et al., 2015). There are several project management tools that can be used for documenting technical debt, e.g. Hansoft, Jira, Redmine (Saraiva et al., 2021). The teams in the company have access to the management tool **Jira** [5] for tracking their work, but not all respondents said that they use this tool (25 out of 26 respondents are using Jira).

Vathsavayi and Systä, 2016 found that the fixing of bugs, implementing new features, and the refactoring of technical debt must be prioritized in the same process. For this prioritization process to be a success, having the technical debt in the same backlog as everything else can be helpful. The same goes for

[5]https://www.atlassian.com/software/jira

security issues. Security work is often not prioritized as it is viewed as less urgent or that it does not produce visible value (Rindell et al., 2019). Having security debt in the same backlog as everything else can make it easier to work with (as it is then not as easily forgotten). There are different ways of **labelling** the technical debt when it is documented (Xavier et al., 2020). Most of the respondents that mentioned labelling technical debt spoke of using the label "NFR" (non-functional requirement). Additionally, labels that said where the technical debt was identified was also added. For security debt it was mentioned that a "security" label was added, often together with the "NFR" label. All tasks added to the backlog is the work that should be done during the next period of time. The people adding the items to the backlog and deciding the order of the items directs the development of the product (Sedano et al., 2019).

5.3.4 Analysis

Technical debt "*measurement quantifies the benefit and cost of known TD in a software system through estimation techniques, or estimates the level of the overall TD in a system*" (Li et al., 2015, p. 204). During the analysis of technical debt in the company, the approach is that the technical debt is given a **severity score**. This score is then added to the ticket in Jira. Risk is, according to Maymi and Harris, 2019, p. 7, "*the likelihood of a threat source exploiting a vulnerability and the corresponding business impact*". This is also how the teams in the company is calculating the severity ("risk severity" = "risk likelihood" * "risk impact"). Before calculating the severity, the teams use scales that vary from 1-3 to 1-10 when giving the likelihood and impact a value.

Several approaches for measuring the technical debt have been mentioned in the literature. Rios et al., 2018 identified 17 approaches, where one of them is cost-benefit analysis and Li et al., 2015 identified 6 approaches for measuring technical debt, e.g. human estimation. A number of tools can also be used during the measurement of the technical debt, e.g. Sonar TD plugin (Li et al., 2015). The main tools used by the company (SonarQube, Coverity, and Snyk) all give an estimate of the identified issues criticality.

During **human estimation approach**, the technical debt is estimated based on experience and expertise (Li et al., 2015). 18 respondents mentioned that they **calculate the severity based on their experience** and 6 respondents said that if the technical debt is reported by a tool, then they take the **score produced by the tool** under consideration. The **cost-benefit analysis** can also be used for analysing the technical debt. This cost-benefit approach provides an analysis of whether or not the repayment of the technical debt is beneficial in regards to the expected interest (Seaman et al., 2012). When calculating the risk, or in the company's case "risk severity", there is always uncertainties that exist, as likelihood is a measure of **uncertainty** (Aven & Renn, 2009). Aven and Renn, 2009, p. 1 also explained that "*risk refers to uncertainty about and severity of the consequences (or outcomes) of an activity with respect to something that humans value*". Calculating the severity provides an indication to how much the technical/security debt will affect the assets that the company value and should also indicate the level of uncertainty. Just like with technical debt, a severity score can also be calculated for the security debt tickets in Jira. When it comes to security debt, it is particularly important with uncertainty awareness because of the potentially large consequences directly related to the system's security.

5.3.5 Monitoring

Technical debt "*monitoring watches the changes of the cost and benefit of unresolved TD over time*" (Li et al., 2015, p. 204). Monitoring the technical debt in the company includes measuring the **SonarQube metrics** and the calculated **severity scores**. Some of the approaches proposed in Li et al., 2015 include planned checks, monitoring technical debt with a focus on quality attributes, and threshold-based approach.

The monitoring that the company employ makes it possible to notice if the amount and severity of the technical debt is increasing or decreasing. This type of monitoring is similar to **planned checks**. During this approach, the identified technical debt is measured regularly and the changes are tracked (Li et al., 2015). Additionally, the company approach can be compared to the proposed **threshold-approach**: getting notified if specific technical debt quality metrics are not met (they do not reach the specified threshold) (Li et al., 2015). This kind of monitoring of technical debt can also work for security debt as there is a need to know if the teams have reached their wanted security goal (is the wanted threshold for security met?). The approach that **monitors technical debt with a focus on quality attributes** can also be looked at in connection to security debt. As security debt have been found to have a connection to a system's security level, it can be said that this approach can work here as well (monitor the change in the quality attribute security) (Li et al., 2015). **Information security continuous monitoring** is defined as "*maintaining ongoing awareness of information security, vulnerabilities, and threats to support organizational risk management decisions*" (Dempsey et al., 2011, p. 1). This is also in line with the arguments for uncertainty awareness above. Security debt should be monitored just like technical debt in order to see how things change, e.g. the environment (Kruchten et al., 2012), (Maymi & Harris, 2019, p. 1086).

5.3.6 Communication

Technical debt "*communication makes identified TD visible to stakeholders so that it can be discussed and further managed*" (Li et al., 2015, p. 205). The company's technical debt process has one listed approach for guiding this activity; **communicating the risks related to the technical debt to the stakeholders**. Communication is one of the activities that are continuously carried out during the entire management of technical debt (Rios et al., 2018). Approaches such as having a technical debt dashboard, backlog, and a list of the technical debt can be used to communicate technical debt to the stakeholders (Li et al., 2015). In the company process it is not specified how the teams should communicate the technical debt risks, only that it is something they should do. According to 7 respondents, communicating technical debt to stakeholder is important and according to 8 respondents, communicating security debt both inside and outside the team is essential. Respondent 1 explained that communicating security debt to the stakeholders is important as it is something that impacts everyone.

Both technical debt and security debt is usually **listed in Jira** and is added to the **backlog** when it is decided that the debt shall be repaid. **Filtering on the labels** added by the respondents makes it possible to view the technical debt and security debt that has been registered. This is in line with having a technical/security debt list and/or dashboard that displays the debt and their respective scores as explained by Li et al., 2015. Communication between the security experts in the team and stakeholders is a must when fixing security issues (Thomas et al., 2018). As this is the case, the team's security engineer should communicate (e.g. through tools or verbally) the security debt to stakeholders.

5.3.7 Prioritization

Technical debt "*prioritization ranks identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases*" (Li et al., 2015, p. 204). During the planning of technical debt, the company approach is: **prioritization based on the given severity score**, is done. Here the prioritization of the technical debt items are set based on the severity. Just like with the tool AnaConDebt (Martini & Bosch, 2017), the result from the calculation is easy to understand, is comparable to other values, and indicates that technical debt should be repaid next. Lenarduzzi et al., 2021, p. 7 mentioned five "*themes illustrating different prioritization aspects*":

1) internal software quality
2) the productivity of the software practitioners
3) the correctness of the software
4) cost-benefit analysis
5) a combination of approaches

Two additional approaches for prioritizing technical debt are the portfolio approach, and the analytic hierarchy process (Rios et al., 2018). Seaman et al., 2012 mentioned a **combination** of these approaches (theme 5 - combination of approaches) for prioritizing the technical debt. The portfolio approach (Guo & Seaman, 2011) and analytic hierarchy process (Alves et al., 2016) both provide a way to rank the technical debt items according to what is the most beneficial moving forward.

For security debt, comparing severity scores can be helpful during the prioritization. The reason for this is that it can be important to repay the security debt items that is found to be the most severe. This is due to the fact that having security debt makes the system more open to attacks because the system has not reached the security goal due to the security debt. Izurieta et al., 2018 mentioned in their study that using the Common Weakness Scoring System (CWSS) method can help with the prioritization of technical debt items related to security. Additionally, using **risk assessment** for prioritizing the security debt is also a possibility (Rindell et al., 2019; Rindell & Holvitie, 2019). Risk assessment "*is a method of identifying vulnerabilities and threats and assessing the possible impacts to determine where to implement security controls*" (Maymi & Harris, 2019, p. 101). Using this kind of approach for prioritizing the security debt can impact the security debt's viewed importance. This is in line with research on securitization understood as applying a rhetoric of threat to issues in order to increase security priorities (Buzan et al., 1998).

5.3.8 Repayment

Technical debt "*repayment resolves or mitigates TD in a software system by techniques such as reengineering and refactoring*" (Li et al., 2015, p. 205). The last activity is repayment. During this activity the company specified that there should be **reserved capacity (20%) for repaying technical debt**. Very few respondents spoke of a specific percentage for technical debt repayment but most of the respondents repay technical debt as part of their daily work (one of the respondents said that as long as the technical debt do not disturb them they will not repay it). Codabux and Williams, 2013 found that using 20% of the Potentially Shippable Increment (PSI) on repayment was an effective way to manage the technical debt. On the other hand, Yli-Huumo et al., 2016 studied how software development teams manage their technical debt, where only two of the eight studied cases mentioned 20% as part of their technical debt repayment. This might indicate that not all teams are working with a given percentage as only a few respondents in this thesis mentioned a percentage for technical debt repayment and only 2 out of 8 of the studied cases by Yli-Huumo et al., 2016 mentioned percentages.

A number of ways to conduct the repayment of technical was found. Some of them are refactoring, reengineering, repackaging, rewriting, and bug fixing (Li et al., 2015). None of these technical debt repayment approaches were specified in the company's technical debt process which might indicate that the teams can do the repayment how they see fit. **Partial refactoring for architectural technical debt** is another approach for technical debt repayment (Rios et al., 2018). During this approach, the technical debt is partially refactored, as complete refactoring is not possible and little to no refactoring can result in a development crisis. Partial refactoring then leads to the most amount of refactoring (Martini et al., 2015).

Respondent 11 was the only respondent that mentioned a specific percentage for repaying security debt. It is a general understanding among the respondents that **security issues are fixed first** and this include security debt. How they choose to do the repayment is up to the teams. Group 5 from table 4.1 holds the respondents that think that security debt does not have straight fixes, meaning that they are not easy to fix. An approach for this kind of security debt can for example be partial refactoring as explained by Martini et al., 2015.

5.3.9 Summing up of RQ2

Following the 8 mentioned activities can provide a structured way of managing the security debt. It can be interpreted from the discussion that having specific security related approaches, or approaches where the security perspective can be added, will benefit the management of security debt. Adding a security perspective to the discussed technical debt approaches focus attention on the security aspects of the debt. Rindell and Holvitie, 2019 explained that "*security debt can be directly integrated into existing technical debt management frameworks and tools with few technical adjustments*". A strategy for this is security risk management (Rindell & Holvitie, 2019), understood as "*the process of identifying and assessing risk, reducing it to an acceptable level, and ensuring it remains at that level*" (Maymi & Harris, 2019, p. 93). It is also worth mentioning the security self-assessment mentioned by some respondents, security self-assessment, mostly related to identification.

To summarize, the main points in this section are:

- Following the 8 mentioned activities from the technical debt process can provide a structured way of managing security debt.
- The approaches used during the technical debt process can also be used during the security debt process by adding a security perspective.
- Having a greater focus on security related approaches are required for

the management of security debt.

5.4 RQ2.1 How is security debt prioritized compared to other kinds of technical debt, feature development, bug fixing, etc.?

The focus of this sub-research question, RQ2.1, is finding out how security debt is prioritized compared to other kinds of technical debt, feature development, bug fixing, etc..

The process of prioritizing technical debt is about choosing what technical debt will be repaid next and what technical debt will be postponed (Li et al., 2015). Repaying all technical debt in a system at once is not possible due to the limited amount of resources allocated to technical debt repayment (Fernández-Sánchez et al., 2017). So, a prioritization takes place, where the calculated severity score plays a significant role.

The prioritization of technical debt repayment and security work
12 respondents explained the importance of prioritizing the fixing of security issues and 6 respondents mentioned that security issues could disrupt sprints due to them needing fixing. On the other hand, security work is often explained to be under-prioritized as it is viewed as less urgent or that it does not provide any visible value (Rindell et al., 2019). This is also often the case for technical debt. The repayment of technical debt is not usually something that impacts the system's external behaviour, but rather something that affects the internal system quality (Fernández-Sánchez et al., 2017). Because of this, teams rather use their resources on feature implementations and bug fixes (Guo et al., 2016).

Prioritization of security debt
11 respondents provided examples of **cases/scenarios where the repayment of security debt was not prioritized**. Respondent 1 spoke of prioritizing production issues over security debt and respondent 9 mentioned having security solutions that would require third-party changes (where the third party is not willing to change), and situations where the security debt have a lower severity score than other work in the backlog. Simply put, in

these cases security debt is something that did not get on top of the prioritization because of more pressing issues or that the available solution is very hard to do. Repaying the principle cost (the cost in order to refactor the system to remove the debt (Avgeriou et al., 2016)) is then not worth it as the interest (the additional cost due to the debts existence (Avgeriou et al., 2016)) generated by the debt is less than the repayment of the principle. Martini and Bosch, 2016 provided a formula ($CPrincipal/TInterest$) for calculating whether or not the repayment of the debt is convenient or not. This provides evidence that security debt would follow a similar prioritization approach as technical debt.

The respondents said, as explained earlier, that the severity score added to the security debt tickets in Jira is a factor that plays a central part in the **backlog prioritization**. Besker et al., 2019 pointed out that the refactoring of technical debt often is not prioritized because there is such a large pressure to produce **customer value** and keeping to the deadlines that feature implementation comes first. Several respondents explained that keeping the customers in mind is important. The respondent 5 said that important security fixes must sometimes be put on hold because a customer wants a feature. The reason behind this is that it is no point in fixing the security issues if there are no customers due to lack of functionality. Ribeiro et al., 2017 presented a **multiple decision strategy criteria model** that not only factored in severity but also factored in the customer impact when prioritizing the debt. As respondent 5 pointed out that security fixes sometimes must be postponed for the benefit of other work, e.g. feature implementation, the multiple decision model might be beneficial here as it factors in the impact on customers.

Lenarduzzi et al., 2021 found in their systematic literature review that even though publications describe having a balance between feature implementation and technical debt repayment as important, none of the identified approaches in the literature review specifically address the prioritization between technical debt repayment and feature implementation. The prioritization approaches only focus on which technical debt items will be repaid next. Having that none of the studied prioritization approaches in Lenarduzzi et al., 2021 specifically mentioned prioritizing other development work together with technical debt repayment, the company's severity/priority score can be beneficial here. If software practitioners add a simple/quick scoring to the features that should be implemented and the bugs that needs fixing then these can be compared to the technical debt items and together be prioritized in the backlog.

Backlog prioritization
The PO/SO prioritizes the backlog (Sedano et al., 2019). 12 respondents said that they have good communication with the PO/SO and that they together discuss the prioritization of the backlog. Having this communication can, according to respondent 21, provide different perspectives (e.g. the PO/SO is a reflection of the stakeholders) when doing the prioritization. It is explained by Sedano et al., 2019 that "*the backlog facilitated communication between professionals with different backgrounds: product designers, product managers, and engineers*" (Sedano et al., 2019, p. 207).

Vathsavayi and Systä, 2016 found that the implementation of new features, the refactoring of technical debt, and the fixing of bugs must be prioritized in the same process. Following that different development work mentioned by Vathsavayi and Systä, 2016 should be prioritized in the same process, it can be argued that security debt should also be a part of this process. In order to do proper prioritizations, comparable values should be used for finding where in the backlog the elements should be placed. There are a number of approaches that can be used for the prioritization, e.g. AnaConDebt (Martini & Bosch, 2016), the multiple decision strategy criteria model (Ribeiro et al., 2017), and as proposed by the studied company; severity/priority scores.

To summarize, the main points are:

- Security issues should be taken seriously as they can disrupt Sprints.
- Evidence that security debt would follow a similar prioritization approach as technical debt.
- Having customer value in mind is important and a model that can help prioritize debt with customers factored in is the multiple decision strategy criteria model.
- Adding severity/priority score to all development work, including feature implementation, bug fixing, technical debt, etc., can help with the prioritization as they then have comparable values.
- Communication between teams and PO/SO provide different perspectives that can be beneficial for a good backlog prioritization.

5.5 RQ 2.2 What role does security knowledge play in the different activities of security debt management?

This sub-research question aim to find out to what extent having security knowledge plays a role in how security debt is handled. As shown in table 3.2 there are a number of respondents that have the role of security engineer as well as other roles. I did not find any clear indication that the respondents with the security engineer role explained security debt and its management any different than the respondents that do not have the this role. The respondents mentioned security knowledge in relation to the security debt management activities prevention, identification, evaluation, prioritization, and repayment.

Knowledge is information in context, meaning that there is a connection between knowledge and information (Barnum & McGraw, 2005). The first two activities mentioned by the respondents are prevention and identification. 11 respondents mentioned that security knowledge was needed for being able to **prevent** security debt and 24 mentioned security knowledge for **identifying** these issues. Assal and Chiasson, 2018 found a connection between security knowledge and security integration. In instances with little security knowledge, they expected that the security practices were lax. Having the needed security knowledge in order to identify and prevent security debt due to lax security practices (as explained by Assal and Chiasson, 2018) is therefore important.

Code review is an approach that is used during the prevention and identification of security debt. According to respondents 13 and 14, code reviews are conducted to catch security issues in the code. Having the required knowledge is important in this approach. This is also explained by Freire et al., 2020 with a technical debt view: lack of expertise in the team can hinder the prevention of technical debt.

Respondent 4 mentioned security debt accumulated due to ignorance or not knowing that there are problems in the system. **Human factors analysis** (Li et al., 2015) can be used to prevent technical debt. This approach can also be applied to security debt; having a team culture where the accumulation of security debt due to ignorance or indifference is reduced.

What then about evaluation? One respondent stated that **evaluating** the security debt cannot be done correctly without security knowledge. This has implications also for prioritization, **prioritizing** without knowledge of security are just guesstimates. The last activity is **repayment**. 22 respondents mentioned this activity, where the answers were divided in two; the ones that view security knowledge as important in all cases and those who think security knowledge is context dependent.

Several reasons for the importance of security knowledge have been presented. Equally important for security debt management is awareness of lack of knowledge.

Lack of security knowledge
There are two descriptions regarding lack of knowledge in the context of technical debt presented in chapter 2. Tamburri et al., 2013 describes a sub-optimal community where they do not have the needed knowledge and therefore have to equalise the debt by for example reverse-engineering the lost knowledge. The other approach is presented by Martini et al., 2015. They found that having a lack of knowledge is a factor/cause for the accumulation of architectural technical debt, a lack of knowledge is divided into four sub-factors: inexperience, lack of domain knowledge, ignorance, and carelessness. Ignorance was also mentioned by one of the respondents as a factor for the buildup of security debt.

Not having the required security knowledge makes it difficult to manage security debt because the various security debt management activities rest on a solid foundation of knowledge. It can also be argued that not being able to execute these activities may result in an accumulation of security debt. Section 5.3 discussed the use of the bug bounty program for identifying security debt. It can be interpreted that teams who heavily rely on the bug bounty program for identifying security debt might indicate that they lack security knowledge.

Sharing security knowledge "*can give a new software security practitioner access to the knowledge and expertise of all the masters*" (Barnum & McGraw, 2005, p. 74). 11 respondents said that knowledge sharing is important. Respondent 15 explained that being able to discuss solutions together can make the developers see things from different perspectives. Sharing security knowledge might also impact how the developers work with the overall security and not just security debt. Maymi and Harris, 2019, p. 1083 explain that when developing both functionality and security together at all phases of the development life cycle, protection will be provided at all the needed layers of the software. The security will then be woven into the core of the software.

In section 5.3 it was explained that communication is one of the activities that are continuously carried out during the entire management of technical debt. Therefore, sharing security knowledge is something that can be part of the communication and therefore be continuous. Thus, it can be interpreted that sharing "*the knowledge and expertise of all the masters*" (Barnum & McGraw, 2005, p. 74) can be a preventative action for security debt accumulation.

The main points from this section are:

- Having a lack of security knowledge have been described to negatively impact the management activities prevention, identification, evaluation, prioritization, and repayment.
- Having a lack of security knowledge can cause security debt to accumulate.
- Sharing security knowledge can be a preventative action for security debt accumulation.

5.6 RQ3 What is the relation between security debt and technical debt?

The aim of this final research question is to find the relation between security debt and technical debt. I will be discussing security vulnerabilities and architectural technical debt when taking a closer look at this relation.

5.6.1 Technical debt, security debt, and security vulnerabilities

Technical debt
How do he respondents think of technical debt? The respondents' technical debt descriptions were divided into four groups (shown in table 4.5). Two of these groups, **deliberate technical debt** and **not deliberate technical debt**, match the factors deliberate and inadvertent presented by Fowler, 2009 in his figure 2.1. Avgeriou et al., 2016 describe that technical debt primarily affects the quality attributes maintainability and evolvability. This is also something that can be seen from the respondents answers. A number of the respondents think of **maintainability** when discussing technical debt. The last of the four groups is in line with the explanation by (Kruchten et al., 2012) that technical debt can due to **change in technology**. This change results in a technology gap that is not due to a wrong choice but rather change in context.

Security debt and technical debt
The description of security debt in group 6 in table 4.1 were divided in three:

- not deliberate security debt
- deliberate security debt
- security debt due to change in technology

This corresponds to three of the four groups created for describing technical debt (table 4.5) as presented above. Just like for technical debt, the **deliberate** and **not deliberate** security debt is in line with the factors deliberate and inadvertent described by Fowler, 2009. Additionally, the **technical gap** is due to the evolution of context, meaning that it is not due to a wrong choice (Kruchten et al., 2012). This can indicate a possible relation between security debt and technical debt.

Furthermore, technical debt can also be understood as "*a way to characterize the gap between the current state of a software system and some hypothesized “ideal” state in which the system is optimally successful in a particular environment*" (Brown et al., 2010, p. 48). The findings in this thesis agree that this kind of explanation can also be applied when it comes to security debt. Security debt has been described by group 7 from table 4.1 to be related to a system's security level (its security goal). According to one respondent, the gap between the "ideal" security level and the current security level is the existing security debt.

Figures 4.3a and 4.3b are modifications of the already presented figures 4.1a and 4.1b, and shows two possible relations between security debt, security vulnerabilities, and technical debt. Siavvas et al., 2019 described that they found a correlation between the vulnerability densities and the technical debt. This relationship can be observed - an increase in technical debt can indicate an increase in vulnerabilities and the other way around (Siavvas et al., 2019). This relationship can be seen in both figures 4.3a and 4.3b. Figure 4.3a shows an overlap between security vulnerabilities and technical debt in the same section where security debt overlaps with security vulnerabilities. Additionally, it is shown that technical debt completely surrounds security debt, just like in figure 4.3b. As explained earlier, the key difference between figures 4.3a and 4.3b is that in figure 4.3b, technical debt is the superset, security debt is a subset of technical debt, and security vulnerabilities are a subset of security debt.

Because of the similarities in the respondents' descriptions of security debt and technical debt, it can be interpreted that security debt can be seen as a type of technical debt and will therefore completely surround security debt as shown in both figures 4.3a and 4.3b. Considering that figure 4.3a is an extension of figure 4.1a it if fair to argue that figure 4.3a best shows the relation between security debt, security vulnerabilities, and technical debt.

5.6.2 Security debt and architectural technical debt

What is the relation between security and architecture? It is described that "*security is best if it is designed and built into the foundation of anything that is build and not added as an afterthought*" (Maymi & Harris, 2019, p. 252). The effect that architecture and security have on each other is described by the respondents from **"somewhat"** to **"a lot"**. Almost half of the respondents mentioned having architectural issues that impacted the security, e.g. moving secrets from one location to another to increase the security/protection. This might indicate a change in the context where different security measures were needed. A respondent pointed out that the needed security is context dependent. Maymi and Harris, 2019 described both that a system's needed security is dependent on the environment in which it exists and that **security controls** should be implemented accordingly. This brings us over to the relation between security debt and architectural technical debt.

"*Security breaches are a recurrent symptom of ATD. Due to the complexity caused by the ATD present in a software-intensive system, inadvertent security flaws can be introduced, leading to the unintentional disclosure of private information to unauthorized parties*" (Verdecchia et al., 2021, p. 14). Most of the respondents that had a clear answer on the relationship between architectural technical debt and security debt said the same thing, that **architectural technical debt could lead to security debt**. Respondent 17 explained that a systems **attack surface** will increase as the architecture gets older and it is not properly maintained. Another cause/factor mentioned as a source of architectural technical debt accumulation is the **evolution of technology** (Martini et al., 2015). Over time, technology becomes more and more obsolete. Using legacy/old components, third-party components, etc. that require the use of old technology is sub-optimal and affects the system. Having these legacy/old components might increase the attack surface and increase a system's vulnerability. Having an increased attack surface is not optimal as this surface is modelled and analyzed for the purpose of narrowing the ways a system can be attacked (Maymi & Harris, 2019, p. 1093). From this we can see evidence that there is a relation between architectural technical debt and security debt.

Respondent 24 said that if a system is not secure then that indicates a bad architecture as one of the quality criteria of good architecture is security. Martini et al., 2015 explained that architectural technical debt "*is regarded as implemented solutions that are sub-optimal with respect to the quality attributes (internal or external) defined in the desired architecture intended to meet the companies' business goals*" (Martini et al., 2015, p. 237). If security is one of the quality attributes that are important to the companies' business goals then not having the needed security (presence of security debt) can contribute to the architectural technical debt. This points to there being a **relation between architectural technical debt and security debt**.

It can be interpreted from the explanations above that by combining the thesis security debt definition and Martini et al., 2015 there is initial evidence that architectural technical debt can cause security debt and vice versa. This relation between the two types of debt require then more awareness during development. It follows from this discussion that the lack of security knowledge cause security debt and that the lack of security knowledge will therefore effect the accumulation of architectural technical debt.

5.6.3 Differences in the technical debt process

Kruchten et al., 2012 found that architectural technical debt is one of the most difficult types of technical debts to identify. It is fair to argue that the technical debt type in question may influence the way technical debt is managed.

Tools is one of the differences in how the types technical debt are handled. The majority of the tools that are used for addressing technical debt are related to design, code, and/or architectural artifacts (Saraiva et al., 2021). This means that tools for other kinds of technical debt such as documentation debt do not have as many tools. Alfayez et al., 2020 on the other hand looked at **approaches** for technical debt management and found that $\approx 71\%$ of the identified approaches for technical debt prioritization works for specific types of technical debt while the rest can be used for any type of technical debt. In addition found Freire et al., 2020 that different preventative approaches also can be used for different types of technical debt. The preventative action "using the most appropriate technology version" is something that can be done for preventing architectural technical debt, while code standardization is for preventing code debt, and documentation debt has the preventative action called "well-defined documentation".

11 respondents pointed out that there should be a difference in how they handle for example architectural technical debt and code debt. **Having approaches for specific types of technical debt might be helpful**, e.g. more security related approaches for the management of security debt. At the same time, having approaches such as code review (used for making sure that the software is of good quality (Maymi & Harris, 2019, p. 1084)) can contribute to managing security debt as well as other types of technical debt, e.g. code debt.

It seems from the discussion presented above that utilizing both specific approaches for specific types of debt and approaches that can used for a number of technical debts might be helpful, in particular to manage the individual needs of the types of debt.

This section discussed the relation between security debt and technical debt. To summarize, the main take away points are:

- Evidence that there is a relation between security debt, security vulnerabilities, and technical debt have been found. This is visualized by figures 4.3a and 4.3b.
- Initial evidence have been found that shows that architectural technical debt can cause security debt and vice versa.
- Different approaches can be used for the management of different types of technical debt. Therefore, having security specific approaches for the management of security debt might be helpful.

5.7 Contributions

The contributions of this thesis is the following:

- An initial definition of security debt have been provided from software practitioner's points of view.
- The relation between security debt and security vulnerabilities have been discussed and two models have been provided.
- A concrete instance of a security debt process have been presented.
- This case study gives evidence to support the fact that security debt can be considered as a type of technical debt.
- Evidence have been found that supports the need for security knowledge in several of the activities of the security debt management and that a lack of security knowledge can be a cause for the accumulation of security debt.
- An initial look at the relation between security debt and architectural technical debt is provided.

Having a common understanding of security debt, just like for other phenomena (e.g. technical debt), makes it easier to communicate. Trying to communicate when people have different definitions in mind makes it hard to be "on the same page". A clear understanding of the relation between security debt and security vulnerabilities can make it easier to understand them and manage them accordingly. This brings us to the security debt process. In order to handle the security debt, a process should be employed in order to make the security debt visible (it is documented, it exists in the backlog, it is prioritized together with the other work, etc.). There are several types of technical debt (Li et al., 2015), having a common understanding of the different types of technical debt, including security debt, can provide for better communication (as technical debt is used as a communication device (Kruchten et al., 2012)). Evidence that lack of knowledge can cause or be a factor in the accumulation of security debt have been found. This can provide a deeper understanding of the value/need for knowledge in different parts of the development. An initial look at the relation between security debt and architectural technical debt is provided. The reason for this is that the complexity in a system caused by architectural technical debt can result unintentional security flaws (Verdecchia et al., 2021).

5.7.1 Recommendations

This thesis provides a starting point for further discussion about security debt. Underneath is the recommendations for practice and research presented.

Recommendations for practice
Based on the contributions I recommend that

- software practitioners in companies should have a common understanding of what security debt is so that there is no misunderstandings about what it is and the implications security debt create.
- companies take inspiration from the findings presented in this thesis regarding the management of security debt. They can create their own security debt process or take inspiration from the one presented (find what fits their company).

- companies have a focus on how architectural technical debt and security debt is connected in order to reduce/avoid potential issues.
- software practitioners have an open channel of communication for sharing security knowledge and experience, especially for security knowledge.

Recommendations for research

- The results in this thesis have provided a security debt definition that can be further studied for finding a formal security debt definition.
- I presented a concrete instance of a security debt process. Further research should be done for finding more relevant activities and approaches for the management of security debt.
- Further research on the management of security debt should be conduced for finding if the same management process can be applied in other companies.
- Evidence that points to there being a connection between security debt and architectural technical debt have been presented. Additional research on this connection would be beneficial in order to find out their specific relationship.
- It would be interesting to find the relation that security debt has to other types of technical debt and not just architectural technical debt.
- The impact penalty points have on the repayment of security debt should be studied for finding if these points positively or negatively impact the repayment of security debt.

5.8 Validity

"*The validity of a study denotes the trustworthiness of the results, to what extent the results are true and not biased by the researchers' subjective point of view*" (Runeson & Höst, 2009, p. 153). The four aspects of validity mentioned for case study research are construct validity, internal validity, external validity, and reliability (Runeson & Höst, 2009, pp. 153–154).

Construct validity
This aspect of validity reflect to what extent the operational measures that are studied really represent what the researcher have in mind and what is investigated according to the research questions" (Runeson & Höst, 2009, p. 153).

During the interviews it was important that me, as the interviewer, and the interviewees understood the concepts we were referring to as the same. Before getting into the main part of the interview I asked the respondents how they would describe technical debt. This gave me the ability to understand how they thought of it and we could continue the interview with the same understanding of technical debt in mind. I also asked the respondents how they would describe security debt, as that is one of the main parts of the thesis.

A threat to the construct validity was when I asked about architectural technical debt. I made an assumption that we understood it the same way. Instead I should have asked how the respondents thought of architectural technical debt, just like I did for technical debt and security debt. Thus, I updated the research questions to take this into account.

Internal validity
"*This aspect of validity is of concern when causal relations are examined*" (Runeson & Höst, 2009, p. 154).

The aim of this study is to provide a deeper understanding of security debt, how it is defined, how it is managed, and its relation to technical debt. The semi-structured interviews opened up for discussion between the respondents and me where it was possible to reach some sort of a common understanding of the topics. I also saw the need to ask more specific follow-up questions on certain topics to mitigate the threat to the internal validity.

External validity
"*This aspect of validity is concerned with to what extent it is possible to generalize the findings, and to what extent the findings are of interest to other people outside the investigated case*" (Runeson & Höst, 2009, p. 154). Runeson and Höst, 2009, p. 154 further describes that the intention for case studies "*is to enable analytical generalization where the results are extended to cases which have common characteristics and hence for which the findings are relevant, i.e. defining a theory*".

The security debt phenomenon has been studied from one company's employees point of view. The findings in this thesis can be understood and applied/used by companies in similar contexts to the studied company (but not other contexts). However, the threat to the external validity is that the difference in context makes it hard for companies in none-similar contexts to apply/use the findings in this study. This is one of the draw-backs of doing qualitative case study research, I only have one point of view, i.e. from one company. On the other hand, this study have provided deep insights into this particular case. Even though the potential of external validity is not high, the company in question should find the findings relevant. However the potential for analytical generalization is higher. Particularly related to the proposed security debt definition and the link between security debt and technical debt. While, the security debt management process is likely to be more context dependent.

Reliability
"*This aspect is concerned with to what extent the data and the analysis are dependent on the specific researchers*" (Runeson & Höst, 2009, p. 154).

A general challenge to the reliability of qualitative research is the researchers influence on the respondents and on the process. It was therefore important for me to have regular discussions concerning my role as a interviewer and how I could avoid influencing the interview setting. Especially concerning my bias and possible influence on the respondents.

Additionally, I tried not to use leading questions during the interviews. But during the transcriptions, I discovered some leading questions. Thus, I had to go back and ask follow-up questions. The process of sending emails to the respondents asking for additional information on questions that were already asked during the interviews increased the reliability of this process. This is due to the fact that these answers were not to the same degree affected by possible bias.

5.9 Limitations

The limitations in this study are as follows:

Time constraint
The interviews I conducted ranged from 45 minutes to 75 minutes. Two things I chose not to dive into due to the time constraint were the use of penalty points and the size of the development teams. Both these factors might have an effect on what work will be prioritized and how much work the teams will be able to finish during each iteration.

Knowledge of architectural technical debt
During the interviews I found that not all respondents seemed to be familiar with architectural technical debt as some of the respondents only responded in the context of architecture.

Data collection method
Due to the pandemic, all interviews were conducted via Zoom. If I had conducted some of the interviews physically I might had gotten additional impressions from the physical surroundings.

Types of technical debt
A number of technical debt types were mentioned in the background but not all were relevant to this study and were therefore not included in the discussion.

Chapter 6

Conclusion

In this thesis, a case study was performed for answering the following research problem: **What is security debt, how is it managed in practice, and what is its relation to technical debt?** To answer the research problem, three main research questions and three sub-research questions were created.

RQ1 How is security debt defined?
This study has proposed the following security debt definition: **security debt is a set of design or implementation solutions that hinder or has the potential to hinder the achievement of a system's optimal/desired/ required security goal**. This definition points to security debt being suboptimal solutions where better solutions exist and the system's security can therefore can be improved. As long as solution to security vulnerabilities exist they are considered as security debt. Thus, these security issues contribute to the hindering or the potential hindering of the achievement of a system's optimal/desired/required security goal.

RQ2 How is security debt managed?
Following the 8 technical debt management activities prevention, introduction, documentation, analysis, monitoring, communication, planning, and repayment provide a structured way of managing also security debt. Additionally, the approaches executed during the security debt management process should have a security perspective. The attention is then on the security aspects of the debt. For prioritizing security debt it was observed that the governing factors are communication, the use of labels, and the use of calculated

severity/priority scores. Adding these severity/priority scores to the development work (e.g. feature implementation and bug fixing) can assist the prioritization as they then have comparable values. Lack of security knowledge have been found to negatively affect several of the security debt management activities and may thereby contribute to the accumulation of security debt.

RQ3 What is the relation between security debt and technical debt? Evidence to support the fact that security debt is a part of the technical debt landscape have been presented. Consequently, the security vulnerabilities that are considered as security debt are then also technical debt. A relation between security debt and architectural technical debt have been observed. Initial evidence shows that architectural technical debt can cause security debt and vice versa. It follows that a lack of security knowledge can then also affect the accumulation of architectural technical debt. There are approaches to the technical debt management that can be used for multiple types of technical debt, while other approaches are specific to one type of technical debt. Therefore, having security specific approaches for the management of security debt might be helpful.

The three main research questions answers the research problem from three perspectives: the definition of security debt, how it can be managed, and how it relates to technical debt. Our society, more specifically interconnected critical infrastructures and functions, is to an increasingly degree dependent on the security of our software systems. Thus, the value of knowledge in the management of security debt cannot be overestimated.

Bibliography

Akbarinasaji, S., Bener, A. B., & Erdem, A. (2016). Measuring the principal of defect debt. *Proceedings of the 5th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering*, 1–7.

Alfayez, R., Alwehaibi, W., Winn, R., Venson, E., & Boehm, B. (2020). A systematic literature review of technical debt prioritization. *Proceedings of the 3rd International Conference on Technical Debt*, 1–10.

Alves, N. S., Mendes, T. S., de Mendonça, M. G., Spınola, R. O., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, *70*, 100–121.

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., & Avgeriou, P. (2015). The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, *64*, 52–73.

Ardi, S., Byers, D., Meland, P. H., Tondel, I. A., & Shahmehri, N. (2007). How can the developer benefit from security modeling? *The Second International Conference on Availability, Reliability and Security (ARES'07)*, 1017–1025.

Assal, H., & Chiasson, S. (2018). Security in the software development lifecycle. *Fourteenth symposium on usable privacy and security (SOUPS 2018)*, 281–296.

Aven, T., & Renn, O. (2009). On risk defined as an event where the outcome is uncertain. *Journal of risk research*, *12*(1), 1–11.

Avgeriou, P., Kruchten, P., Ozkaya, I., & Seaman, C. (2016). Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162) (P. Avgeriou, P. Kruchten, I. Ozkaya, & C. Seaman, Eds.). *Dagstuhl Reports*, *6*(4), 110–138. https://doi.org/10.4230/DagRep.6.4.110
Keywords: coding tools and techniques, design tools and techniques, management, metrics, software engineering

Barnum, S., & McGraw, G. (2005). Knowledge for software security. *IEEE Security & Privacy*, *3*(2), 74–78.

Besker, T., Martini, A., & Bosch, J. (2019). Technical debt triage in backlog management. *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 13–22.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., et al. (2010). Managing technical debt in software-reliant systems. *Proceedings of the FSE/SDP workshop on Future of software engineering research*, 47–52.

Buzan, B., Wæver, O., Wæver, O., De Wilde, J., et al. (1998). *Security: A new framework for analysis*. Lynne Rienner Publishers.

Codabux, Z., & Williams, B. (2013). Managing technical debt: An industrial case study. *2013 4th International Workshop on Managing Technical Debt (MTD)*, 8–15.

Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative, and mixed methods approaches* (5th ed.). SAGE Publications.

Cruzes, D. S., & Dyba, T. (2011). Recommended steps for thematic synthesis in software engineering. *2011 international symposium on empirical software engineering and measurement*, 275–284.

Cunningham, W. (1992). The wycash portfolio management system. *ACM SIGPLAN OOPS Messenger*, *4*(2), 29–30.

Dempsey, K., Chawla, N. S., Johnson, A., Johnston, R., Jones, A. C., Orebaugh, A., Scholl, M., & Stine, K. (2011). Information security continuous monitoring (iscm) for federal information systems and organizations. https://doi.org/10.6028/NIST.SP.800-137

Eisenberg, R. J. (2012). A threshold based approach to technical debt. *ACM SIGSOFT Software Engineering Notes*, *37*(2), 1–6.

Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., & Gorton, I. (2015). Measure it? manage it? ignore it? software practitioners and technical debt. *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 50–60.

Fernández-Sánchez, C., Garbajosa, J., Yagüe, A., & Perez, J. (2017). Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. *Journal of Systems and Software*, *124*, 22–38.

Fowler, M. (2009). *Technical debt quadrant*. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

Freire, S., Rios, N., Mendonça, M., Falessi, D., Seaman, C., Izurieta, C., & Spınola, R. O. (2020). Actions and impediments for technical debt prevention: Results from a global family of industrial surveys. *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 1548–1555.

Greenwood, D. J., & Levin, M. (2006). *Introduction to action research: Social research for social change*. SAGE publications.

Grenness, T. (1997). *Innføring i vitenskapsteori og metode*. Tano Aschehoug.

Guo, Y., & Seaman, C. (2011). A portfolio approach to technical debt management. *Proceedings of the 2nd Workshop on Managing Technical Debt*, 31–34.

Guo, Y., Spınola, R. O., & Seaman, C. (2016). Exploring the costs of technical debt management–a case study. *Empirical Software Engineering*, *21*(1), 159–182.

Hanssen, G. K., Brataas, G., & Martini, A. (2019). Identifying scalability debt in open systems. *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 48–52.

ISO/IEC. (2011). https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en

Izurieta, C., Kimball, K., Rice, D., & Valentien, T. (2018). A position study to investigate technical debt associated with security weaknesses. *2018 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 138–142.

Izurieta, C., & Prouty, M. (2019). Leveraging secdevops to tackle the technical debt associated with cybersecurity attack tactics. *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 33–37.

Kruchten, P., Nord, R. L., & Ozkaya, I. (2012). Technical debt: From metaphor to theory and practice. *Ieee software*, *29*(6), 18–21.

Kruchten, P., Nord, R. L., Ozkaya, I., & Falessi, D. (2013). Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt. *ACM SIGSOFT Software Engineering Notes*, *38*(5), 51–54.

Laszka, A., Zhao, M., Malbari, A., & Grossklags, J. (2018). The rules of engagement for bug bounty programs. *International Conference on Financial Cryptography and Data Security*, 138–159.

Lenarduzzi, V., Besker, T., Taibi, D., Martini, A., & Fontana, F. A. (2021). A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software*, *171*, 110827.

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, *101*, 193–220.

Lim, E., Taksande, N., & Seaman, C. (2012). A balancing act: What software practitioners have to say about technical debt. *IEEE software*, *29*(6), 22–27.

Lindgren, M., Wall, A., Land, R., & Norström, C. (2008). A method for balancing short-and long-term investments: Quality vs. features. *2008 34th Euromicro Conference Software Engineering and Advanced Applications*, 175–182.

Martinez, J., Quintano, N., Ruiz, A., Santamaria, I., de Soria, I. M., & Arias, J. (2021). Security debt: Characteristics, product life-cycle integration and items. *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*, 1–5.

Martini, A., & Bosch, J. (2016). An empirically developed method to aid decisions on architectural technical debt refactoring: Anacondebt. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 31–40.

Martini, A., & Bosch, J. (2017). The magnificent seven: Towards a systematic estimation of technical debt interest. *Proceedings of the XP2017 Scientific Workshops*, 1–5.

Martini, A., Bosch, J., & Chaudron, M. (2015). Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study. *Information and Software Technology*, *67*, 237–253.

Maymi, F., & Harris, S. (2019). *Cissp all-in-one exam guide, eighth edition* (8th ed.). McGraw-Hill Education.

McGraw, G. (2004). Software security. *IEEE Security & Privacy*, *2*(2), 80–83.

Merriam-Webster. (n.d.). Metaphor [visited on 14-05-2022]. *Merriam-webster.com dictionary*. https://www.merriam-webster.com/dictionary/metaphor

Miles, M. B., & Huberman, A. M. (1994). *Qualitative data analysis: An expanded sourcebook*. sage.

Miles, M. B., Huberman, A. M., & Saldana, J. M. (2013). *Qualitative data analysis: A methods sourcebook* (3rd ed.). SAGE Publications.

Nord, R. L., Ozkaya, I., Kruchten, P., & Gonzalez-Rojas, M. (2012). In search of a metric for managing architectural technical debt. *2012 Joint Working*

IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture, 91–100.

Pérez, B., Castellanos, C., Correal, D., Rios, N., Freire, S., Spınola, R., Seaman, C., & Izurieta, C. (2021). Technical debt payment and prevention through the lenses of software architects. *Information and Software Technology*, *140*, 106692.

Ribeiro, L. F., Alves, N. S. R., Neto, M. G. D. M., & Spınola, R. O. (2017). A strategy based on multiple decision criteria to support technical debt management. *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 334–341.

Rindell, K., Bernsmed, K., & Jaatun, M. G. (2019). Managing security in software: Or: How i learned to stop worrying and manage the security technical debt. *Proceedings of the 14th International Conference on Availability, Reliability and Security*, 1–8.

Rindell, K., & Holvitie, J. (2019). Security risk assessment and management as technical debt. *2019 International Conference on Cyber Security and Protection of Digital Services (Cyber Security)*, 1–8.

Rios, N., de Mendonça Neto, M. G., & Spınola, R. O. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology*, *102*, 117–145.

Robson, C. (2002). *Real world research: A resource for social scientists and practitioner-researchers*. Wiley-Blackwell.

Runeson, P., Host, M., Rainer, A., & Regnell, B. (2012). *Case study research in software engineering: Guidelines and examples*. John Wiley & Sons.

Runeson, P., & Höst, M. (2009). Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering*, *14*(2), 131–164.

Saraiva, D., Neto, J. G., Kulesza, U., Freitas, G., Reboucas, R., & Coelho, R. (2021). Technical debt tools: A systematic mapping study. *Proceedings of the 23rd International Conference on Enterprise Information Systems*, 88–98.

Schreiber, A., Sonnekalb, T., & von Kurnatowski, L. (2021). Towards visual analytics dashboards for provenance-driven static application security testing. *2021 IEEE Symposium on Visualization for Cyber Security (VizSec)*, 42–46.

Scott, C., & Medaugh, M. (2017). Axial coding. *The international encyclopedia of communication research methods*, *10*, 9781118901731.

Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., & Vetrò, A. (2012). Using technical debt data in decision making: Potential decision approaches. *2012 Third International Workshop on Managing Technical Debt (MTD)*, 45–48.

Sedano, T., Ralph, P., & Péraire, C. (2019). The product backlog. *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE)*, 200–211.

Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D., Anicic, N., & Gelenbe, E. (2019). An empirical evaluation of the relationship between technical debt and software security. *9th International Conference on Information society and technology (ICIST), 2019*.

Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., & Tzovaras, D. (2020). Technical debt as an indicator of software security risk: A machine learning approach for software development enterprises. *Enterprise Information Systems*, 1–43.

Silva, M. C. O., Valente, M. T., & Terra, R. (2016). Does technical debt lead to the rejection of pull requests? *arXiv preprint arXiv:1604.01450*.

Sneed, H. M. (2014). Dealing with technical debt in agile development projects. *International Conference on Software Quality*, 48–62.

Tamburri, D. A., Kruchten, P., Lago, P., & van Vliet, H. (2013). What is social debt in software engineering? *2013 6th International Workshop on Cooperative and Human Aspects of Software Engineering (CHASE)*, 93–96.

Thomas, T. W., Tabassum, M., Chu, B., & Lipford, H. (2018). Security during application development: An application security expert perspective. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12.

Tom, E., Aurum, A., & Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, *86*(6), 1498–1516.

Vathsavayi, S. H., & Systä, K. (2016). Technical debt management with genetic algorithms. *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 50–53.

Verdecchia, R., Kruchten, P., & Lago, P. (2020). Architectural technical debt: A grounded theory. *European Conference on Software Architecture*, 202–219.

Verdecchia, R., Kruchten, P., Lago, P., & Malavolta, I. (2021). Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software*, *176*, 110925.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., & Wesslén, A. (2012). *Experimentation in software engineering*. Springer.

Xavier, L., Ferreira, F., Brito, R., & Valente, M. T. (2020). Beyond the code: Mining self-admitted technical debt in issue tracker systems. *Proceedings of the 17th International Conference on Mining Software Repositories*, 137–146.

Yli-Huumo, J., Maglyas, A., & Smolander, K. (2016). How do software development teams manage technical debt?–an empirical study. *Journal of Systems and Software*, *120*, 195–218.

Appendix A

Interview guide

Interview guide security debt

Consent

- Quick introduction about the study and what we will be talking about.
- Documenting consent before the interview starts.

Introduction

- Can you tell me a little bit about the project you are working on?
  - Team size
  - Project duration
  - Hats/roles

Technical debt (TD)

- How do you think of technical debt?
  - Do you have an example of a technical debt item?

Technical debt process and architectural technical debt (ATD)

- Do you have a process for technical debt that you follow?
  - Is it a formal or informal process?
- Do you follow the same technical debt process for all the different types of technical debt?
  - When do you do it differently?
- Have you had an architectural issue that had a security impact?

Security debt (SD) and security debt process

- When you hear the term security debt what do you think of?
  - Do you have an example of a security debt item?
- How should the security debt process be?
- How is security debt different from security vulnerabilities?
  - Are there security vulnerabilities that are not security debt and vice versa?
- What effect do you think ATD and SD have on each other?
  - SD a symptom of ATD?

Security debt prioritization

- How is security debt prioritized?
  - Security knowledge in the team?
  - Marking of security debt for backlog prioritization?
  - Severity, impact and likelihood?
  - In relation to other kinds of technical debt, bug fixing, etc.?

Finish interview

- Send email to interviewee with additional contact information in case there is something they want to add or if they have questions.
- Can I send you an email if I need some clarifications or have some additional questions?
- This is great, thank you for participating!