Chapter 1

Introduction

1.1 Overview

Designing and developing software-intensive products and services is a significant undertaking. The decisions made during the design and development of software-intensive products and service are affected by the fast pace of software development. Organisations and development teams often compromise to cope with demanding deliveries and deliverables by making sub-optimal decisions [7, 97]. Dealing with such challenges by acting fast to change and by making sub-optimal decisions might jeopardise the value of the delivered product, environment, development process, and the artefacts which are involved, such as source code and documentation [22]. Management of such artefacts is crucial for efficient quality control in the engineering, maintenance, and evolution of software-intensive products and services.

To elaborate and move towards a more holistic view of assets with inherent value for the organisation [137], we first need to have a thorough definition of what assets are. The term asset is still too often used carelessly without putting much thought into what an asset eventually constitutes and, consequently, how the assets should be managed. Despite the central role of assets in software engineering, management, and evolution, there is no clear consensus about what assets are when dealing with software-intensive products and services. Therefore, a structured terminology of the concepts related to asset management such as "value" or "value degradation" is required. We need to create the body of knowledge to provide us with an overarching view of the assets that are involved in the process of designing and developing software-intensive products

and services. After providing a clear definition for assets, there is a need for designing a well-defined vocabulary and taxonomy of assets in Software Engineering.

Defining the concepts used in asset management, such as value and value degradation, enables us to move towards studying assets' value degradation. Measuring the degradation of assets enables us to monitor the assets during the inception, planning, development, evolution and maintenance. It will also lead to better management of the assets in organisations. Measuring the degradation of individual assets requires a specific investigation of its type, i.e., the measurements might differ for each type in its context. Lastly, existing measurements can be used collectively to assess the degradation of assets.

This work consists of four articles that address two objectives. This thesis focuses on:

- First, formalising the concept of asset management by providing the definitions of the terminology of the related concepts. We aim at creating a taxonomy of assets that allows us to provide a structured and organised way of classifying assets that are important for organisations during the inception, planning, development, evolution, and maintenance of software-intensive products and services.

- Second, investigating how the assets in software-intensive products and services can be evaluated and measured, where we start by narrowing the topic to the assets related to source code and architecture to examine the methods we can use to assess such assets by evaluating and measuring them.

The next chapter describes the background of artefacts in software engineering, technical debt, and assets (Section 1.2). We continue by describing the research questions and contributions (Section 1.4). We summarise the research methodology utilised to answer the research questions of this licentiate thesis in Section 1.5 and provide a summary of the results of each study in Section 1.6. Finally, at the end of this chapter, we summarise the future work planned for the rest of the thesis work (Section 1.7).

The rest of this thesis includes chapters build upon included publications as follow. Chapter 2 describes the positioning of the concept of assets providing definitions for the terminology used in this context. Chapter 3 provides the taxonomy of assets that are created based on a literature review and focus groups. Chapter 4 delves into the degradation of code by investigating how various development activities impact the accumulated code technical debt. Chapter 5 provides a method for prioritising the code technical debt items by investigating their survivability.

1.2 Background and related work

In this section, we will cover the background for *artefacts in software engineering*, *technical debt*, and *assets in software engineering*.

1.2.1 Artefacts in Software Engineering

Software artefacts are important entities for the inception, planning, development, evolution, and maintenance of software-intensive products and services. Therefore, they should be organised and managed meticulously. The definition and organisation of software artefacts have a significant influence on software development. Software artefacts have been defined as "documentations of the results of development steps" [22], work results, or work products produced, modified, maintained and used by a sequence of tasks that have value to a role [22, 80]. Considering the many different software engineering areas that deal with various artefacts, there is a need for an organised structure for their management [22]. This especially evident when we consider the descriptions and documentation of large organisations that deal with software-intensive products and services. The existing definitions of the artefacts do not provide a structure and are mostly monolithic [116], i.e., there is no organised, hierarchical structure for artefacts that illustrates their characteristics and properties and further the relations among them. Poorly structured artefacts are difficult to reuse, and the evolution of such monolithically structured artefacts is cumbersome [107].

There are artefacts that only have importance for a specific role but do not have value from a broader organisational perspective. Therefore, they are either not persistent and inherently transient [137], or their impact on the whole process is marginal. We can distinguish artefacts relevant to the organisation if they participate in the development process over time, and we need quality assurance mechanisms to control their quality. The definitions of artefacts presented in the literature do not distinguish artefacts that have an inherent value for the development organisation and are subject to degradation over time from the artefacts that do not have any value for the organisation.

1.2.2 Technical Debt

Technical debt (TD) was introduced by Cunningham in 1992 to describe the consequences of making sub-optimal decisions made to gain immanent goals [27]. The TD metaphor has been extended to include more than source-code-related artefacts, and it is currently one of the interesting topics in software engineering [97]. The metaphor has been studied in numerous articles in academia, and it has been incorporated as

a concept widely used and discussed by industrial practitioners as it gains popularity [59]. At the moment, TD is one of the critical issues in the software development industry, and if it remains unchecked can lead to significant cost overruns [14].

Alongside the metaphor, practitioners and researchers started to implement practices to deal with such sub-optimal solutions in Technical Debt Management (TDM) activities [7, 43]. These activities include prevention, identification, monitoring, measurement, prioritisation, and technical debt repayment [97].

Eventually, the TD metaphor grew to cover other software development activities by introducing technical debt types, such as documentation and testing technical debt (documentation debt and test debt) [7]. After the introduction of TD types, researchers began to investigate these classifications. One of the earlier papers providing a TD classification is the work of Tom et al. [117]. There have been other secondary and tertiary studies on the topic ever since, such as the work of Li et al. [72] and Rios et al. [97].

As mentioned before, the TD metaphor originally used the financial terminology of debt and interest in the context of software development, and it focused, mainly, on the source code, i.e., mostly code, design, and architecturee [7]. Therefore, other aspects of software development, such as social and organisational aspects, were not addressed. Eventually, the metaphor was extended, e.g. test debt and documentation debt. Therefore, other aspects of TD did not receive the same amount of attention until recently, e.g., requirements debt and documentation debt [97]. However, these aspects have been studied in their own related literature.

Though TD types have been under investigation by researchers in various articles, there is a lack of studies investigating the permeation and co-occurrence of TD types [7]. For example, if and how a growing code TD will impact test TD. Moreover, there is a need to investigate TDM activities [7]. There are secondary studies that investigate TDM activities with different perspectives by, for example, focusing on tools or strategies. However, there is still a lack of unified analysis aligning these different perspectives [97]. Finally, most studies disregard the evolutionary aspect of TD [33, 95]. Whereas studying TD's evolution is a more appropriate approach [20].

The assets (i.e., artefacts that we are interested in) are subject to technical debt, which makes the research on TD a good input area to find assets. However, we do not delimit ourselves to the artefacts that are discussed in TD. Technical debt is part of asset degradation defined in [137]. We can use technical debt for an initial attempt to evaluate and measure assets' degradation. Moreover, the TD literature is concerned with different aspects of software development. The TD metaphor has been used as an umbrella term to include all the negative consequences that can occur in the software development process. TD has received a growing interest in academia and the industry.

Moreover, the TD topic has been studied in other venues and research for a considerable time.

1.3 Empirical Background and Cases

Most of the research done in this licentiate thesis is empirical in its nature. Empirical methods, such as case studies and experimentation, are used to investigate software engineering phenomena and have become popular among researchers [30]. Empirical software engineering aims at investigating scientific evidence on phenomena related to software development, operation, and maintenance is applied by many researchers in the field [30]. The empirical research depends on the collaboration of academia and industry where scientific evidence is created by investigating industrial cases and it is used to support practitioners.

The research done in this thesis is based on empirical evidence collected and synthesised in collaboration with five companies in the construction machinery, communication technology industry, banking and financial services and open-source systems. The research partner companies are Ericsson, Fortnox, Time People Group, Qtema, and Volvo CE. All the collaborating companies work on software-intensive product development and are involved in ongoing research projects (SERT[1] and SHADE). Table 1.1 summarises the collected data with reference to Figure 1.1 – overview of the methodology.

Finally, the research presented in Chapter 5 is based on the investigation of 31 open-source systems that are from the Apache Foundation. We have extracted the data from 31 systems with the largest amount of source code (over 10,000 lines of code (LOC)) written in Java.

1.4 Research Questions and Contributions

This licentiate contributes to the software engineering field by providing a different perspective on the concept of assets in software engineering and, more specifically, on asset management and asset degradation.

This section outlines the research questions (RQ) that steered this thesis and the research contributions.

[1] See `www.rethought.se`.

Table 1.1: Data Sources Used in The Thesis

Study	***Stage*** **Data Source** **[Data Type]**
Chapter 1	*Research Clarification*
Vision Paper	-
	[–]
Chapter 2	*Descriptive Study I*
Literature Review	Academic Peer Reviewed Articles
	[Qualitative Data – Nine Peer-Reviewed articles]
Focus Groups	Focus groups with the participation of five Companies (Ericsson, Fortnox, Time People Group, Qtema, and Volvo CE)
	[Qualitative Data - Collected Notes]
Chapter 3	*Prescriptive Study*
Case Study	Industrial Case
	[Quantitative Data - Extracted from archival information of version control and static code analysis tool]
Chapter 4	*Prescriptive Study*
Sample Study	Two industrial case and 31 OSS Cases
	[Quantitative Data - Extracted from archival information of version control and static code analysis tool]

1.4.1 Research Questions

Two research questions are investigated in this thesis:

RQ1. What are the assets important for organisations during the inception, planning, development, evolution, and maintenance of software-intensive products and services?

Developing software-intensive products and service requires utilisation of many various "assets", denoting the inherent value of assets to the organisation. There are terms and concepts used in software engineering that are often intermixed and used inconsistently. This is evident when examining the literature on technical debt, and it is observed while working with our industrial partners. The role of assets in software engineering, management, and evolution is paramount. Yet, there is no clear consensus as to which artefacts are assets when dealing with software-intensive products and service. Therefore, a structured terminology is required to clearly define the distinctive characteristics of assets that allows us to differentiate assets from artefacts and that allows us to frame the asset management concepts, such as "value" or "value degradation". Furthermore, a taxonomy of assets can help capture a vast body of knowledge by allowing us to cluster and classifying different assets involved in software-intensive products and service.

To answer this research question, we started with positioning the concept of assets and defining the related concepts (Chapter 2). Later, we created a taxonomy of assets while considering both academic and industrial input. The taxonomy was created based on a literature review and four focus groups (Chapter 3).

RQ2. How can we evaluate and measure the degradation of assets important for organisations during the inception, planning, development, evolution, and maintenance of software-intensive products and services?

After clarifying the concepts and terminology and identifying the assets, the next research question deals with the evaluation and measurement of asset degradation. This evaluation aims at understanding how we can capture the degradation of assets and eventually leads to the creation of methods and frameworks to manage asset degradation. Given that by defining the concepts and creating the taxonomy, we covered a wide range of software development, evaluating and measuring all assets requires tremendous effort and much time. Therefore we started by investigating a subset of assets. We decided to begin with assets

related to the source code and architecture. There are well-defined metrics for source-code- and architecture-related assets, and data are abundant both from industrial projects and open source systems. Moreover, starting with a smaller set of assets will make it easier to create and evaluate methods that can eventually be used for other assets.

To answer this research question, we started by examining the existing metrics and methods to evaluate source-code- and architecture-related assets. Initially, we examined how various development activities impact the accumulation of TD by conducting an industrial case study. Later, we investigated a method to analyse the survivability of code technical debt items, more specifically, bugs, code smells, and vulnerabilities, to investigate their longevity.

1.4.2 Research Contributions

This licentiate thesis contributes to the software engineering field by introducing the concept of asset management and its body of knowledge. Also, we investigate existing evaluation methods and metrics as a starting point to create methods or frameworks to manage assets and their degradation. The detailed contributions are:

C1. Defining the concept of assets in the development of software-intensive products and services and creating the initial taxonomy that identifies different assets *(RQ1)*.

The findings contribute to the main objective of this thesis by providing a clear, standardized definition of the concept of asset, asset management, and asset degradation. Moreover, we have identified and classified the assets and types of assets in a taxonomy that can be used as a base for future studies and exploration of assets, their characteristics, the concept of degradation, and different types of degradation.

C2. Understanding the impact of bug fixing, refactoring, and development of new features on source-code-related assets (*RQ2*).

Software-intensive products and services products are constantly maintained, and as part of their evolution, their size and complexity increases [65, 66].. The evolution of software products is bound to the various activities that happen during the development. Be it be due to the development of new features, improvements of the code quality through refactoring, or fixing the bugs in the system. Such activities impact the degradation of the code and or test bases (either positive, contributing to its remediation or negative, contributing to the degradation).

We examined an industrial case to understand the effect of different activities on code technical debt, which is closely related to its degradation. The results can be a starting point to help us better understand these activities' effect source-code degradation.

C3. Using a statistical method for evaluating the survivability of certain source-code problems that can help prioritise the efforts to deal with their degradation (*RQ2*).

We examine the survivability of source-code-related problems, namely bugs, code smells, and vulnerabilities, by utilising survival analysis[2]. We have performed a sample study inspired by the work of Tufano et al. [121, 122]. The results of such analysis can be integrated with other methods to eventually help prioritise the activities that deal with the degradation of source-code-related assets.

1.5 Research Design

We used quantitative methods and quantitative methods to address research questions stated in Section 1.4 (See Figure 1.1). The methods selected to conduct the studies included a literature review, focus groups, case study, sample study, and various quantitative methods and analysis techniques, i.e., statistical analyses such as survival analysis [83]. The rationale for selecting the mentioned methods, together with a short description for each method, is provided in the rest of this section.

1.5.1 Research Methods

We used a qualitative approach to answer the first research question of the licentiate thesis. What assets are important for organisations during the inception, planning, development, evolution, and maintenance of software-intensive products and services? The individual methods used for answering the research question are described below.

Focus Group

The objective of the focus groups was to identify the important assets for organisations from their perspectives. The author of the licentiate thesis participated in two of the four focus groups. The two other focus groups were held by other researchers. In those

[2]Survival analysis is a statistical method the analyses the probability of occurrence of events [83]

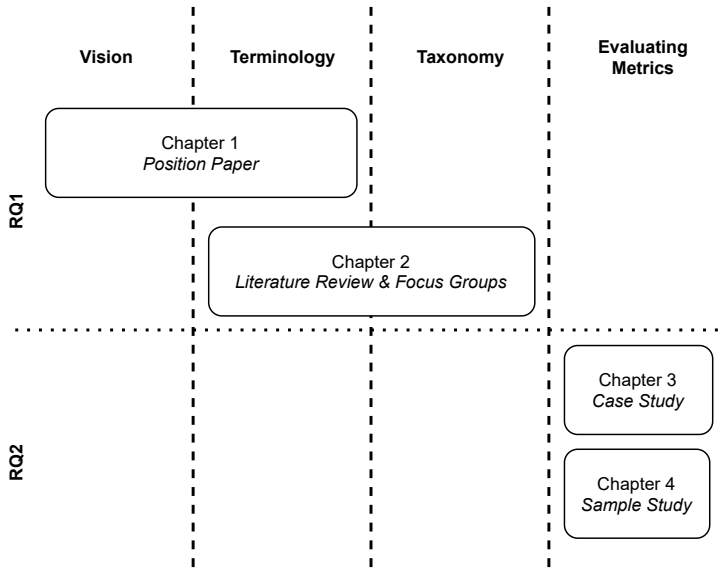

Figure 1.1: Research Methodology Overview.

cases, the author has used the collected data and performed a secondary analysis of the data collected by other researchers [98].

A Focus group is a method of collecting qualitative data through a groups interview that consist of demographically similar people [84]. Focus groups can be on open topics or specific topics with the guidance of a session moderator. Focus groups are used to extract the perception of the participants on the topic. If the population attending the focus groups are large enough and representative of the population, the findings can be extrapolated and generalised. However, we are not aiming for statistical generalisability.

Group size and the moderator's involvement are the important elements when conducting focus groups [84]. While larger groups are suitable for open topics, smaller groups with the involvement of the session moderator provide a better opportunity for each participant to discuss their views and experience [84] and therefore are better suited for highly specialised topics.

An alternative for collecting similar data is conducting individual structured interviews or surveys. Individual structured interviews or surveys lack the multiplicity of views and do not provide a holistic view and the aggregated knowledge of the partici-

pants. To identify the important assets (RQ1) from the industrial perspective, we opted to perform focus groups since they are relatively straightforward to perform, less time consuming, and accommodate a larger sample size. More importantly, "focus groups elicit a multiplicity of views and emotional processes within a group context." [40].

We have followed the labelling guidelines provided by Saldaña [104] to label the collected data from focus groups. The focus groups included:

1. The introduction session where participants introduced themselves.
2. The presentation sessions where the mediating researchers presented the topic.
3. The focus groups where participants discussed their opinion on the topic.

The details of the focus groups are described extensively in Chapter 3.

The analysis of the data gathered from focus groups together with the literature review led to the creation of the taxonomy. The results show that the amount of identified organisational and managerial assets are similar to other aspects of software development. Therefore, studying the degradation of such assets are as important as the rest.

Literature Review

Reviewing the literature has become one of the foundations of evidence-based software engineering [31]. Systematic literature reviews, systematic mapping studies, and tertiary studies help present the current knowledge and findings of a specific topic. To investigate the terminology used in the area of assets and asset management, we reviewed secondary and tertiary research papers that classify assets and provide definitions of the term. We collected the initial set of papers using a search string and completed the set following the snowballing guidelines provided in [129]. There are alternative literature synthesis methods, such as *expert review* or *ad hoc literature selection* [54].

We performed a literature review on the topic of technical debt. At the start of the work, the focus of the research was on the topic of technical debt. After the focus groups with the collaborating partner companies, we realized that there is an interest to have a holistic view of assets rather than focusing solely on a narrow subject of technical debt. The technical debt (TD) metaphor deals with sub-optimal solutions that have a long-term impact on the system and its development. We assume that if TD is important to study for a certain artefact, it might indicate that the artefact is important for the organization the same way as assets are important for organization.

The details of the literature review are described extensively in Chapter 3. We have performed the snowballing technique, and we have followed the guidelines of Wohlin [129]. The literature review resulted in the creation of an asset matrix based on input

from academia. We were able to extract and synthesize the important assets that are mention in the reviewed literature. The results of this step were combined with the data collected from the focus groups to create the taxonomy of assets.

> We have used a quantitative approach to answer the second research question of the licentiate thesis: (RQ2) How can we evaluate and measure the degradation of assets important for organisations during the inception, planning, development, evolution, and maintenance of software-intensive products and services? The individual methods used for answering the research question are described below.

Case Study

A case study is a method for collecting data for a specific purpose of a study which is performed on project, activities, and assignments [99, 135]. Case studies are often observational, and the collected data is often investigated using qualitative analysis and quantitative analysis [99, 135]. A case study is "an empirical enquiry that investigates a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident." [135, p. 13].

We chose to perform case study research to investigate architecture- and source-code-related assets. The objective of our case study was to explore the impact of bug fixing, refactoring, and new development on the accumulation of TD in an industrial context. The study was conducted on a specific software component. We selected this case because of the availability of the case. We could retrieve certain information from this case that is hard to retrieve from similar cases from the industry.

An alternative method for investigating the phenomenon is action research. In action research, the researcher engages with members of the organisation – in our case, industrial partners – to diagnose problems and produce solutions [84]. We opted to perform a case study since: (i) the goal of the study and what it investigates, i.e., research questions, are clear from the beginning; (ii) data is collected in a consistent manner; (iii) the research questions are answered by inferences from data; (iv) and the threats to the study's validity can be systematically addressed [30, 100].

The results of the case study show how various development activities impact the degradation of source-code-related assets. And the longevity of certain issues that lead to the degradation of such assets. The results can be used to better understand the degradation of source-code-related assets and can provide insights when managing assets and dealing with their degradation.

Sample Study

A sample study is a method that is performed over a certain population of actors, and it aims to achieve generalizability [111]. Sample study is one of the popular methods in software engineering research, and it is frequently used to study large sets of software development projects, in particular OSS projects [111]. We chose to perform a sample study to investigate the survivability of code technical debt items in an industrial system and 31 open-source systems from the Apache Foundation.

Given the fact that sample studies might not provide depth of insight, alternative methods for investigating the phenomenon can be used. Such studies can be field studies or case studies that require more specific context [111]. However, field studies and case studies have their limitations. The limitations of the field studies are the following: It cannot provide statistical generalizability [111]. There is no control over events, and there is low precision on measurements [111]. And case studies are difficult to conduct on a large scale.

1.5.2 Limitations and Threats to Validity

The validity of the research discusses the credibility and dependability of the results by considering to what extent the results are not biased [100]. The threats to validity for each study is described in details in each respective chapter. However, we recognise that this thesis has its general limitations.

The work presented in this thesis is based on empirical data, and the relevance of the results is limited to the context of the research. We tried to strengthen the work presented in this licentiate thesis by involving partner companies from different industries. We tried to mitigate this threat by involving different roles from the partner companies. The research work has been conducted during 2019, and the results represent that period.

Internal validity is not a significant concern to this licentiate thesis since we do not prescribe any particular practice, method, or framework to address the research questions in the licentiate thesis. Instead, we focus on contextualising the challenges and problems presented in this licentiate thesis.

Threats to external validity refer to the generalisability of the results and to what extent the results are interesting to people outside of the investigated cases [100]. Ivarsson and Gorschek [50] present four aspects to evaluate research relevance, namely subjects, context, scale, and research methods. The research methods used in this licentiate thesis mostly investigate industrial context, i.e., real-life context. The data used in this work is primarily collected from industrial cases. The subjects participating in this research are system architects, product owners, project managers, team leaders, enter-

prise architects, solution architects, and many more with the first-hand experience from the industry. Therefore, their insights are on par with the general population of industry professionals.

The context and scale of our research are organisations that deal with software-intensive products and services. A representative sample would allow for the generalisation of the results. An understanding of the whole population is needed to precisely characterise the representativeness of the sample [89]. However, we cannot have a precise understanding of the population due to the sheer number of organisations dealing with software-intensive products and services. Throughout this thesis, we use convenience sampling and cannot guarantee that all kinds of organisations are represented.

Reliability refers to the extent that the data and analysis are dependent on the researchers. When conducting qualitative studies, the threat to validity is the replicability of the results and the process [70]. The studies included in this licentiate thesis are designed by the authors, and the interpretations of the results might be affected by their biases. We tried to mitigate this threat, all the data collected for the studies and the results of the studies are jointly reviewed among the authors of each study.

1.6 Summary of the Results

In this section, the main results of the studies are briefly described. The results are mapped to the research questions and show how they help to answer the research questions.

1.6.1 Assets in Software Engineering

The first research question in this licentiate thesis includes positioning the concept of assets that are important for organisations during different development stages of software-intensive products and services. The terminology used in the concept of assets such as asset management, value, value degradation, and asset degradation is also defined in this step. Finally, a taxonomy of assets is created as an initial effort to summarise the existing body of knowledge and a method of communication using consistent terminology.

The created taxonomy contains the perspective of both academia and industry. It is important to mention that the taxonomy is created in a way that it can be extended as the field matures and evolves. There are eight major types of assets identified in the taxonomy, with overall 57 assets. The major types are:

- *Product-Requirements-Related Assets*

- *Product-Representation-Related Assets*
- *Development-Related Assets*
- *Verification-and-Validation-Related Assets*
- *Operation-Related Assets*
- *Environment-and-Infrastructure-Related Assets*
- *Development-Process- /Ways-of-Working-Related Assets*
- *Organisation-Related Assets*

An interesting observation of the results is that there is a balance between the number of assets that are related to the source code / architecture of the software and the assets that are related to organisational and managerial aspects of the development.

Identifying the assets is the first step to study their degradation. The results are important to both researchers and practitioners. The results can steer future research and provide a road map and common terminology. The results can also help the industry to better identify and manage the assets involved in software-intensive products and service.

1.6.2 Towards Characterising the Properties of Assets

A natural progression of the work of this licentiate thesis is to investigate the degradation of the mentioned assets. Given the multitude of assets in the taxonomy, which includes various topics in software engineering research, we decided to narrow our focus and study the assets that are related to source code and architecture.

To understand and characterise the properties of source-code and architecture- related assets, we started by examining the existing metrics and methods used to evaluate such assets. We believe that by better understanding assets' characteristics and by combining the existing methods and metrics, we will be able to monitor and manage the degradation of assets.

The results provided in Chapter 4 and Chapter 5 aim to help us achieve this goal. We examine the impact of different development activities during the development. The results show that, contrary to expectations, different development activities can have an unexpected impact on the source-code-related assets. Furthermore, we examine the survivability of source-code-related issues to understand better whether the persistence of certain types of technical debt issues in the software systems can indicate their degradation.

1.7 Future Work

The long-term objective of this thesis is to contribute to creating a framework to manage various assets or a particular subset of assets that are used in designing and developing software-intensive products and service. As the next steps, we plan the following:

- Gathering feedback and impressions for the taxonomy. Our aim is to present the taxonomy in the industry and gathering feedback from practitioners, and report their impressions.
- A natural progression of this work is also to explore and identify how the industry measures their assets' degradation and what tools they used to measure assets' characteristics and properties. The idea is to present the assets (results from Chapter 3)) to practitioners and collect their input on the metrics and tools used to manage assets and their degradation.
- We intend to investigate a subset of assets presented in this thesis, namely source-code- and architecture-related assets. Focusing on a narrower topic, we will study the degradation of code and architecture. We will investigate the methods and tools that can be utilised to deal with said assets' degradation.
- Finally, we plan to create a framework/method for managing source-code- and architecture-related assets by identifying the degradation and prioritising management activities to deal with their degradation.