Chapter 1

Introduction

1.1 Overview

Designing, developing, and maintaining software-intensive products and services is a complex and significant undertaking with many challenges. There are a myriad of decisions that need to be taken by stakeholders during all stages of software development [106]. The fast pace of software development may lead to sub-optimal decisions that impact the maintenance and evolvability of the software [14].

There are many software artefacts that are created, maintained, and evolved during software development and are essential to the final product or service's quality. The quality of software artefacts that are exposed to continuous change and increasing complexity may decrease unless proper quality assurance is exercised on them [116]. Therefore, it is crucial that software developing organisations be able to identify software artefacts that are involved in the development process [143].

The continuous maintenance and evolution of software impacts the artefacts involved in the development process. The constant use, maintenance, and evolution causes software artefacts to degrade in quality. The degradation of software artefacts needs to be managed since they have value for stakeholders [143]. Furthermore, the degradation of some of these artefacts might have bigger impact on the organisation as they are more relevant, i.e., they are used often over time by the stakeholders in a given organisation.

There are various factors that impact the quality degradation of software artefacts. And there have been many studies investigating software artefacts' quality from different perspectives, for example [3, 64, 127, 167, 214]. Practitioners and researchers use metrics to evaluate the quality of software artefacts. One of the metrics that has become popular both in industry and in academia is technical debt [178]. Technical debt has traditionally been used to measure the consequences of short-term decisions on long-term software development [14, 106]. The technical debt metaphor has been extended and studied by many researchers [178], it has been an interesting topic for both academia and industry, and it has grown from a metaphor to a practice [107].

A lot of research has been carried out regarding specific and well-known software artefacts and how they degrade, for example [118, 128, 137, 178]. While the lack of awareness of software artefacts might not pose significant challenges in developing software-intensive products or services, identifying software artefacts that are used often and whose quality needs to be controlled is necessary for reach more effective artefact management activities. In other words, organ-

isations can invest in exercising quality control activities on artefacts that are used often and are more relevant.

Another aspect related to quality degradation is propagation of degradation among different artefacts, i.e., degradation of an artefact worsening the degradation of other related artefacts. Several studies have investigated the propagation of quality degradation among software artefacts (e.g., [2, 198]). Therefore, raising the awareness and identification of software artefacts with greater value for organisations and their quality degradation has the potential to help organisations to be more successful and efficient when developing software-intensive products or services.

> ***Software assets*** *are those relevant software artefacts that have long-term strategic value —they are intended to be used more than once— and can provide a competitive advantage to the organisation. These may include reusable software components, libraries, and other intellectual property that can be leveraged across multiple projects and products. Proper management of software assets can lead to significant cost savings, increased productivity, and faster time-to-market for new software solutions.*

This thesis summarises the findings of empirical investigations regarding the identification of software assets, that are relevant for the development organisations, and their quality degradation. The contribution and the flow of the research presented in this thesis are illustrated in Figure 1.1. The research gaps G1 and G2 are presented Section 1.3. The research approach using design research methodology (DRM) is presented in Section 1.5. And the thesis contribution are presented in Section 1.6.

The main aim of this thesis is two-fold: i) help organisations understand and identify relevant software assets; and ii) provide empirical evidence illustrating quality degradation of software assets, as well as organisation factors that might exacerbate this quality degradation.

Furthermore, this thesis intends to foster improvements by providing: a taxonomy of relevant software assets and an empirically created model to assess the alignment degree between ownership and contribution.

The rest of the this chapter is structured as follows: Section 1.2 contains relevant background together with a discussion of the related work. The research gaps are presented in Section 1.3. The addressed research questions in this thesis are presented in Section 1.4. The research approach, methodologies used in this thesis, and the threats to validity are presented in Section 1.5. The

contributions of the work is summarised in Section 1.6. Section 1.7 present the discussion followed by conclusions and future work in Section 1.9.

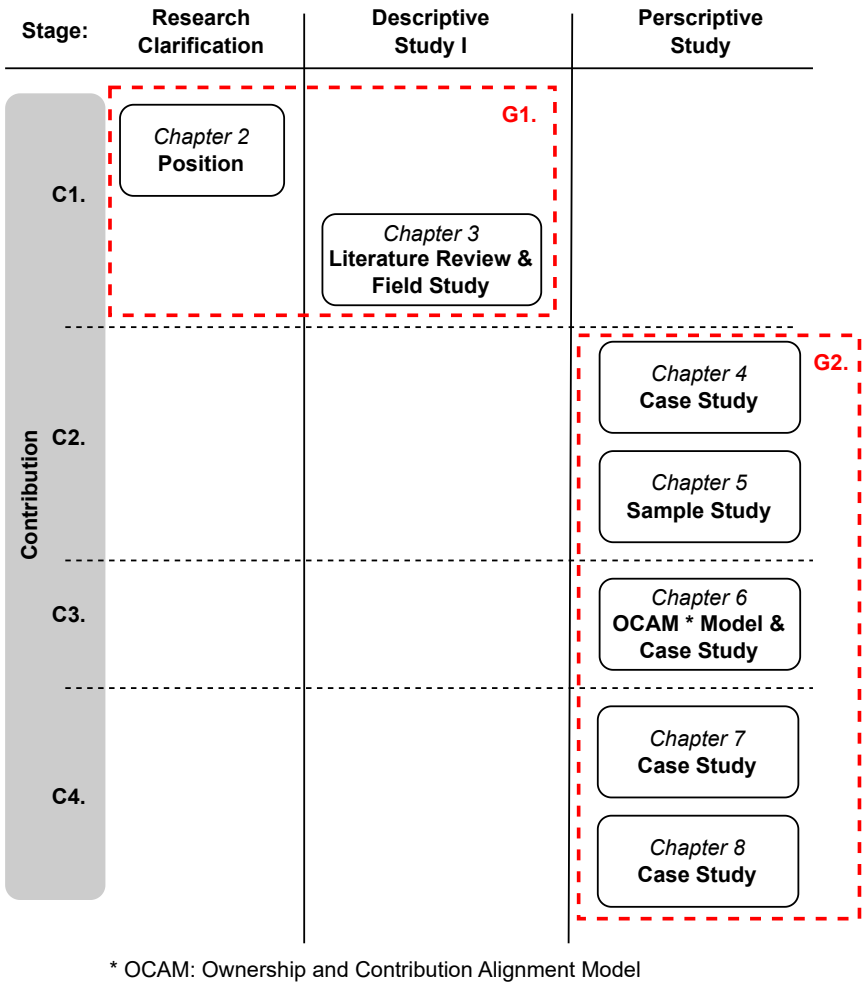

Figure 1.1: Overview of the chapters and thesis contribution. Boxes denote how chapters of this thesis map to the contributions, research gaps, and design research methodology (DRM) framework stages [33].

1.2 Background and Related Work

The section summarises the background for the thesis. The research conducted and presented in this thesis mainly relates to four topics: i) Artefacts in Software Engineering, ii) Software Evolution, and iii) Technical Debt.

1.2.1 Artefacts in Software Engineering

Software artefacts are of high importance to software development. Documents, deliverable, and work products are examples of software artefacts. They have been around since the beginning of software development practices [143]. Software artefacts have been defined in different sources. A software artefact is:

- "Documentation of the results of development steps" [45].
- "A work product that is produced, modified, or used by a sequence of tasks that have value to a role" [141].
- "A self-contained work result, having a context-specific purpose and constitutes a physical representation, a syntactic structure and a semantic content of said purpose, forming three levels of perception" [141].

Understanding software artefacts, how they are structured, and how they relate to each other has a significant influence on software-intensive developing organisations [45]. The need for organising software artefacts structurally is highlighted as the documentation in large-scale systems grow exponentially [45].

Artefacts defined by most of the Software Development Processes (SDPs) are monolithic and unstructured [204]. The content of poorly structured artefacts is difficult to reuse, and the evolution of such monolithic artefacts is cumbersome [190]. Therefore, different SDPs present various models for presenting software artefacts, e.g., the Rational Unified Process (RUP) [103, 104]. There are ways to classify and structure software artefacts based on well-known modelling concepts. Examples of such models are the work of Broy [45] and Silva et al. [190]. Moreover, there are ontologies and meta-models to classify artefacts in specific software development areas (e.g., Idowu et al. [89] Mendez et al. [142], Zhao et al. [232], and Constantopoulos and Doerr [52]).

However, the definitions of artefacts presented in the literature do not distinguish between artefacts that have an inherent value for the development organisation or artefacts that are intended to be used more than once. There is a lack of definition of such artefacts and the terminology around them. Software

developing organisations can improve quality control practises by identifying these important artefacts.

It is important to identify and distinguish relevant software artefacts, i.e., software assets, whose quality needs to be controlled since they have different characteristics and value propositions for software development organisations.

On the other hand, software artefacts are temporary or disposable items that are created during the software development process. While artefacts are essential for managing the development process and documenting the software system, they do not have the same long-term value as software assets.

By identifying and distinguishing software assets from software artefacts, organisations can develop effective strategies for managing their quality degradation. They can prioritise the development and maintenance of assets that provide the most value, while streamlining the management of artefacts to reduce waste and improve efficiency. This can lead to more effective software development practices, better quality software solutions, and improved business outcomes.

1.2.2 Software Evolution

Software evolution refers to the process of changing and adapting software systems over time in response to changing requirements, technologies, and user needs. It is an ongoing process that involves the modification, enhancement, and maintenance of existing software systems throughout their lifecycle.

Software evolution is a natural and necessary aspect of software development [177], as software systems are rarely developed once and then left untouched. Instead, they typically evolve in response to changing requirements, bug fixes, and new features. Evolution can also be driven by external factors such as changes in the underlying technology stack, security vulnerabilities, and user feedback.

Software products and services, due to their nature, evolve [177]. The development process for software-intensive products or services is continuous, i.e., software products or services constantly change to include new features and functionality or to fix bugs or provide improvements [114, 116]. And as the software product or service is changing and growing, it's complexity increases, unless work is done to maintain or reduce it [116]. Therefore, there is a need for constantly maintaining the software to keep the quality satisfactory [114, 116].

The rapid change of software, i.e., the ability to rapidly evolve software, creates challenges for software-developing organisations to develop reliable, high-quality software [145]. Software changes have been studied comprehensively

(e.g., [47, 129]). The literature categorises software changes into *corrective*, *adaptive*, *perfective*, and *preventive* modifications [177]. Such modifications are discussed in the scope of software systems and products and in relation to the degradation of the external functionality.

Software artefacts that are involved in the development of any software-intensive product evolve as new functionality and improvements are implemented in the product or service. The evolution and change of software artefacts implies that their quality will diminish if proper quality control practices are not practised [114, 116]. Therefore, the quality degradation of software artefacts due to their evolution is important to consider when planning management activities.

1.2.3 Technical Debt

Software artefacts are impacted by the evolution of software product or service, i.e., due to change and maintenance. Evolving software artefacts need to adhere to a certain level of quality or standards. When a stakeholder does not keep the quality of any artefact to its standard, they incur 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 [55]. The TD metaphor has been extended to include more than source-code-related artefacts, and it is currently one of the topics receiving more attention in software engineering [14, 178]. 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 [178]. 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 increases [30, 106, 178].

TD is recognised as one of the critical issues in the software development industry [29, 178]. TD is "pervasive", and it includes all aspects of software development, signifying its importance both in the industry and academia [106]. The activities that are performed to prevent, identify, monitor, measure, prioritise, and repay TD are called Technical Debt Management (TDM) [14, 85] and include such activities as, for example, identifying TD items in the code [212, 218], visualising the evolution of TD [63, 162], evaluating source code state [123, 212], and calculating TD principal [7, 56].

As the TD metaphor was extended to include different aspects of software development, various TD types were introduced [14], e.g., requirements debt, test debt, and documentation debt. The introduction of different types of TD has led researchers to attempt to classify the different types and categories of TD.

One of the earliest classifications of TD is the work of Tom et al. [206]. Other secondary and tertiary studies have been performed to summarise the current state of TD and TD types, e.g., by Lenarduzzi et al. [118], Rios et al. [178], and Li et al. [128].

The metaphor has been mainly *operationalised* by researchers and practitioners for source code, i.e., code, design, and architecture [14]. The organisational, environmental and social aspects of TD have not received the same amount of attention [178]. Moreover, TD types are often studied in isolation, not considering how one TD can be contagious to other assets and the permeation to other TD types [14]. For example, when the TD in code grows, a valid question would be whether and how it impacts the TD in tests and the extent of such impact.

Technical debt management activities have been investigated in several secondary studies with different perspectives, some focusing on tools, others on strategies, but there is still a lack of unified analysis aligning these different perspectives [178].

TD is generally studied in the current state of the software, and it is understudied with regards to the evolutionary aspects of TD, i.e., studying TD on a "snapshot" of a system is not enough [68, 171]. Therefore, a more appropriate approach to studying TD is to study its evolution [43].

There are articles on the implications of the impact of low ownership of a module on code quality (e.g., [170]). However, to best of our knowledge, there has not been any study on the impact of team ownership and contribution alignment on the faster accumulation of TD. Finally, there are limited studies on the impact of organisational factors, such as team structure or ways-of-working on the faster accumulation of TD.

1.3 Research Gaps

The existing literature has investigated and defined software assets. There is a need to organise and structurise software artefacts [142, 204] that are valuable for software developing organisations, i.e., software assets. Currently, there is no structured way with which an organisation can define and identify the valuable artefacts. The first gap addressed in this research is:

> ***G1.*** *Lack of a definition of software assets, asset degradation, and a taxonomy of such assets.*

Consequently, lack of a definition may lead to fuzzy, unclear, and or even contradictory understanding of the essential concepts involved such as software

assets, which is essential to further study the change of quality of assets over time, i.e., asset degradation. Providing empirical evidence, especially from industry(e.g., [2, 198, 210]), solidifies the understanding of said assets. However, there is a need of more empirical evidence supporting asset degradation [178]. Understanding different organisational factors that might have an impact on the faster degradation of assets is the next step to achieving this goal. The second gap addressed in this thesis is:

> ***G2.*** *The need to provide empirical evidence from industrial cases supporting and illustrating the concept of asset degradation, and how different organisational factors might impact the faster accumulation of assets' quality degradation.*

This thesis fills the existing gaps by employing empirical research using quantitative and qualitative analysis of data collected from industry. The next section provides the research questions address in this thesis and how they address the mentioned gaps.

1.4 Research Questions

This thesis addresses the following research questions:

> **RQ1.** What is asset degradation in software engineering?

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 not used consistently. 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 services. Therefore, a structured terminology is required to clearly define the distinctive characteristics of assets that allows us to differentiate assets and that allows us to frame the asset management concepts, such as "asset degradation".

To answer this research question, we started with positioning the concept of assets and defining the related concepts, i.e., asset degradation and asset management (Chapter 2). We have identified the different types of asset degradation namely deliberate, unintentional, and entropy. And we have discussed

their differences with examples. We discuss the propagation of degradation on assets. Degradation propagation is the influence of assets' degradation to other dependent assets. Finally, we created a taxonomy of assets while considering both academic and industrial input. The taxonomy was created based on a literature review and a field study (Chapter 3).

This thesis explores the ideas and concepts introduced and coined during the research. The research conducted to answer is exploratory and it is conducted to find evidence and indications to support the assumptions. The quality degradation of certain software assets have been studied before as mentioned in Section 1.2. However, there is a growing interest to study the degradation of assets and its implications through case studies, sample studies, and field studies.

> **RQ2.** What factors impact the degradation of assets when developing software-intensive products or services?

After clarifying the concepts and terminology and identifying the assets, this research question deals with understanding the impact of different factors on asset degradation, starting with organisational factors. The thesis aims at understanding the organisational factors that might impact degradation of certain assets. Understanding and identifying such factors is necessary to create methods and models to effectively manage asset degradation. However, the creation of such models is outside of the scope of this thesis.

Given that by defining the concepts and creating the taxonomy, we covered a wide range of software development body of knowledge, 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 can be retrieved 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. We performed studies to investigate asset degradation first to identify different factors related to the degradation of assets, and second to evaluate their impact on asset degradation using industrial data. These case studies are presented in Chapters 4, 5, 6, 7, and 8. Additionally, we created a model to measure the degree of alignment between the ownership and contribution that led to study-

ing such alignment's impact on code degradation, i.e., by faster accumulation of technical debt.

1.5 Research Approach

This section provides an overview of the research approach in this thesis. The section contains a summary of the research methods and empirical data sources used in this thesis. This section is concluded by presenting the limitations and threats to validity of the thesis.

The research presented in this thesis utilises both quantitative and qualitative research methods and data. Therefore, a mixed-methods research approach [54] is used in order to answer the research questions mentioned in the previous section.

Most of the research done in this thesis is empirical in its nature. Empirical methods, e.g., case studies, aimed at investigating scientific evidence on phenomena related to software development, operation, and maintenance [66]. Empirical research relies on the collaboration of academia and industry, where scientific evidence is created by investigating industrial cases, and it is used to support practitioners. A brief introduction to the different research methods presented in this thesis is provided next; an overview of the different methods, along with the contributions of individual studies, is depicted in Table 1.1.

The research presented in this thesis uses the design research methodology (DRM) framework as a guideline to systematically design the research [33]. The DRM framework is proactively used to structurise the content of this thesis.

The DRM framework consist of four stages namely *Research Clarification*, *Descriptive Study I*, *Prescriptive Study*, and *Descriptive Study II* [33]. Figure 1.2 illustrates the stages, the link between stages, basic means, the main outcome from each stage, and the mapping of the chapters presented in this thesis to the DRM framework. The bold arrows between the stages illustrate the main process flow, the light arrows the many iterations. The *research clarification* stage is where the researchers find evidence or indications to support their assumptions. The goal of the second stage, i.e., *descriptive study I*, is to help researchers determine which factors to address and obtain a better understanding of the phenomena before moving to the next stage. In the *prescriptive study*, researchers use their previously found understanding about the phenomena to correct and elaborate on their initial description in the first stage. And finally, in the *descriptive study II* stage, the researchers investigate the impact

of the support and its ability to realise the desired situation. There are seven main types of design research within the DRM framework.

The research presented in this thesis follows the second type of design research, i.e., *Comprehensive Study of Existing Situation* which contains the first three stages of the DRM framework. The rest of this section presents an overview of the stages.

1.5.1 Stage 1: Research Clarification

This stage is dedicated to find indications or evidence that supports the assumptions that lead to the research goal. During this stage, the researchers provide an initial description of the existing and the desired situation. Chapter 2 presents the Research Clarification stage in this thesis.

Table 1.1: Overview of the research approach, data collection methods, and analysis methods presented in this thesis.

	Chapters						
Research Approach	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Literature Review		•					
Field Study		•					
Case Study			•		•	•	•
Sample Study				•			
Data Collection Method	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Snowballing		•					
Focus Group		•			•	•	•
Archival Data Retrieval			•	•	•	•	•
Analysis Method	**2**	**3**	**4**	**5**	**6**	**7**	**8**
Statistical Analysis			•	•	•	•	•
Thematic Analysis		•				•	

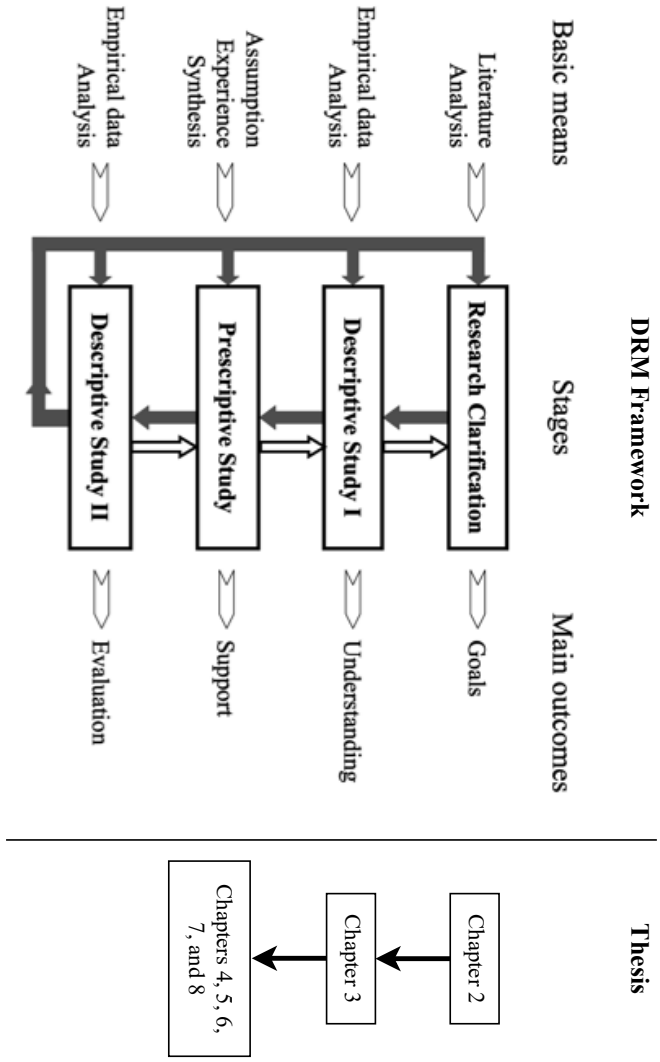

Figure 1.2: The design research methodology (DRM) framework [33] and thesis chapters.

1.5.2 Stage 2: Descriptive Study I

Following the previous stage, i.e., having a clear goal and focus, this stage is dedicated to reviewing the literature and observing the existing situation. During this stage, the researcher obtains a better understanding of the existing situation and provides the basis for the next stage of the DRM framework. Chapter 3 presents the Descriptive Study I stage in this thesis. Literature review and field study are used to obtain the goal of this stage.

- **Literature Review** A literature review is a comprehensive and critical analysis of published research literature, books, and other sources related to a particular topic or research question. It aims to provide an overview of existing knowledge and research on a specific subject, identify gaps in the literature, and highlight areas for further research [100].

  Reviewing the literature has become one of the foundations of evidence-based software engineering [67]. Systematic literature reviews, systematic mapping studies, and tertiary studies help present the current knowledge and findings of a specific topic. To investigate the terminology of software artefacts, we performed an analysis of secondary and tertiary research papers that classify technical debt types and provide definitions of the terms. We collected the initial set of papers using a search string and completed the set following the snowballing guidelines provided in [220]. There are alternative literature synthesis methods, such as *expert review* or *ad hoc literature selection* [100].

  We performed a literature review using snowballing (see Chapter 3 for more details) on the topic of technical debt as it is used to discuss the degradation of software artefacts. The technical debt (TD) metaphor deals with sub-optimal solutions that have a long-term impact on the system and its development. Investigating technical debt helps us reason about their degradation. Finally, we assume that if TD is important to study for a certain artefact, it might indicate that the artefact is important for the organisation the same way as assets are important for organisation.

  The analysis of secondary literature presented in this thesis belongs to stage two of the DRM model, i.e., Descriptive Study I. It is the follow up to the position paper presented in Chapter 2. We performed the analysis of secondary literature to find about the current state of research related to technical debt types as a proxy to assets.

- **Field Study**

  Field studies are the research that are conducted in a real-world, natural setting that study a specific phenomena. Field studies are appropriate for the cases where the researchers will not change any of the variables, i.e., there is no deliberate change or intervention from the researchers [196]. Researchers conduct field studies when they aim to develop a deep understanding of a phenomena in its setting, e.g., the organisation studied in this thesis.

  The objective of the field study was to identify the valuable artefacts, i.e., assets, for organisations from their perspectives. The data collected during the field study is used to perform a secondary analysis [179]. To identify the assets from the industrial perspective, we opted to perform a field study since it facilitates the study of the phenomena in its context and the researchers aimed to understand *what is going on* and *how things work*, i.e., exploration of the phenomena. We have followed the labelling guidelines provided by Saldaña [184] to label the collected data from the field study.

  We performed a field study (see Chapter 3 for more details) with five companies to identify and collect software assets from industry's perspective and gain an initial description of the current situation (Stage two of the DRM model). The field study is complementary to the systematic literature review. The collected data led to the creation of a taxonomy of software assets.

1.5.3 Stage 3: Prescriptive Study

This stage is dedicated to increase the understanding of the current investigated situation and elaborate on the desired situation. In this stage the researchers examine various different factors that impact the situation. During this stage the researchers examine different factors that impact the current situation. Chapters 4, 5, 6, 7, and 8 present case studies and a sample study used to obtain the goal of this stage.

- **Case Study**

  A case study is a research methodology that involves an in-depth investigation of a particular individual, group, organisation, event, or phenomenon. It is an empirical inquiry that is often used in social sciences, business, education, and other fields to examine complex real-life situations and to

gain a deep understanding of the underlying factors and dynamics at play. In a case study, the researcher collects and analyses a data sources such as interviews, observations, documents, and archival data to generate detailed and nuanced descriptions of the case being studied. The ultimate goal of a case study is to develop insights, theories, and recommendations that can inform future research, practice, and decision-making in the relevant field [180, 225].

Case studies are often observational, and the collected data is often investigated using qualitative analysis and quantitative analysis [180, 225]. 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" [225, p. 13]. We chose to perform five industrial case studies to investigate architecture- and source-code-related assets. The objectives of the case studies were to explore and understand the factors that might impact quality degradation using technical debt as a proxy for code and architectural degradation in industrial context. The studies were conducted on a specific software components. We selected the cases by convenience, because of the availability of the cases and collaborating companies. We could retrieve certain data and information from these cases 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 implement interventions as potential solutions [84]. We opted to perform a case study since: (i) the nature the study was still exploratory; (ii) data is collected in a consistent manner; (iii) the research questions are answered by inferences from data; (iv) the data collected allows us to plan interventions, but they have been yet implemented in the studied organisations.

The results of the case studies help us understand the impact of quality degradation in different cases and components while considering different factors and each components' context. The results can be used to understand the degradation of source-code-related assets and can provide insights when managing assets and dealing with their degradation. We conducted the case studies (See Chapters 4, 6, 7, and 8) to understand impact of the degradation of certain assets and elaborate on the different factors that impact their degradation (stage three in the DRM framework, i.e., Prescriptive Study).

- **Sample Study**

  A sample study is a method that is performed over a certain population of actors, and it aims to achieve generalisability [196]. 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 [196]. 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 [196]. However, field studies and case studies have their limitations. The limitations of the field studies are the following: It cannot provide statistical generalisability [196]. There is no control over events, and there is low precision on measurements [196].

  We performed a sample study to investigate the survivability of code technical debt items in industrial and open-source settings. The sample study presented Chapter 5 belongs to stage three of the DRM model, i.e., Prescriptive Study. And it is conducted to understand the impact of the degradation of certain assets.

1.5.4 Empirical Data Sources

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 described companies work on software-intensive product development and are research partners in research projects (SERT[1] and SHADE). Table 1.2 summarises the empirical data source and data type used in this thesis and maps them to each chapter.

[1] See www.rethought.se

Table 1.2: Data sources and types used in this thesis.

Chapter	**Empirical Data Source [Data Type]**
Chapter 3	Academic Peer Reviewed Articles [Qualitative Data – Tertiary study of nine literature reviews]
	Five Companies (Ericsson, Fortnox, Time People Group, Qtema, and Volvo CE) [Qualitative Data - 413 Collected Notes and Statements]
Chapter 4	One Industrial Case - One system (2 MLOC) [Quantitative Data - Extracted from archival information of version control and static code analysis tool]
Chapter 5	Two Industrial Cases (2 MLOC) - 31 OSS Cases (3 MLOC) [Quantitative Data - Extracted from archival information of version control and static code analysis tool]
Chapter 6	One Industrial Case - 267 Components (15 MLOC) [Quantitative Data - Extracted from archival information of version control and static code analysis tool]
	One Company - Six Participants [Qualitative Data - Statements from Focus Group used for validation of results]
Chapter 7	One Industrial Case - Five Components (40.5 KLOC) [Quantitative Data - Extracted from archival information of version control and static code analysis tool]
	One Company - Six Participants [Qualitative Data - 30 Collected Notes and Statements and 10 Pages of Focus Group Transcriptions]
Chapter 8	One Industrial Case - 10 Components (132 KLOC) [Quantitative Data - Extracted from archival information of version control and static code analysis tool]
	One Company - Six Participants [Qualitative Data - Statements from Focus Group used for validation of results]

1.5.5 Validity

This section presents the validity of the results presented in this thesis. The studies conducted in this thesis have limitations which are discussed separately in each corresponding chapter. Here, we will focus on the *generalisability*, *replicability*, and *reliability* of the results of the thesis. The discussed validity is based on the criteria provided by Runeson et al. [181] and Yin [225].

- **Generalisability:** Generalisability is the extend to which the findings of the research can be generalised outside of the investigated cases and be applicable in other situations [181]. Ivarsson and Gorschek [93] present four aspects to evaluate research relevance, namely subjects, context, scale, and research methods. The methods used in this work investigate industrial cases in real-world context. The collected data is from the investigated systems and components. The subjects participating in this research are from different roles including: system architects, product owners, project managers, team leaders, enterprise 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 [163]. However, we cannot have a precise understanding of the population due to the sheer number of organisations dealing with software-intensive products and services. We tried to mitigate this thread by including different organisations and components in this research. Throughout this thesis, we use convenience sampling and cannot guarantee that all kinds of organisations are represented.

- **Replicability:** Replicability is the ability of obtaining the same results by other researchers using the data collected with different setups [38, 39]. The majority of the results presented in this thesis are context-specific and related to the companies investigated for each study. Therefore, a thread to validity of this work is the problem of replicating the same results in such context. In order to mitigate and limit this threat, the authors have provided meta-models, source-code for analysis and data collection, and detailed description of the collected data to provide sufficient information for replication of the results.

- **Reliability:** Reliability refers to the extent that same findings and conclusions can be obtained by different researchers if they follow the same procedure as the original study [225]. When conducting qualitative studies, the threat to validity is the replicability of the results and the process [124]. The work presented in this thesis are designed, conducted, and replicated by the authors. The results might be affected by their personal experience and biases. We tried to mitigate this threat by involving more than one author when collecting and analysing the data and drawing conclusions.

1.6 Research Contributions

This thesis contributes to the software engineering field by defining assets, the terminology in asset management, and types of asset degradation. A taxonomy of assets in software engineering is created and empirical evidence from industrial cases is presented. The thesis presents a model to evaluate degree of ownership and contribution alignment for a given component. Moreover, the thesis presents industrial case studies conducted to provide evidence on asset degradation by assessing factors that might impact its faster accumulation. Figure 1.1 summarises the contributions of the thesis. Each chapter of the thesis is mapped to a contribution, research gap, and a DRM stage. The details of the thesis contribution are presented below:

C1. Definition of assets in software engineering, the terminology in asset management, and types of asset degradation.

The first contribution of this thesis is providing the definition of assets in software engineering and its and related terminology such as asset degradation and its types. Moreover, a taxonomy of assets is created to identify an initial set of software assets. The thesis finding motivate the investigation on software assets and asset management in software engineering. This contribution is related to *G1.* and includes the papers in Chapters 2 and 3.

C2. Empirical evidence on asset degradation.

The second contribution of the thesis is providing empirical evidence from industrial cases illustrating asset degradation. This thesis provides four case studies and a sample study to investigate and provide empirical evidence on asset degradation. Despite many existing literature on quality

degradation on technical debt, evidence from industrial cases are scarce. The investigated cases in this thesis are from different settings and different companies. Each study considers different factors and illustrates how each factor might impact specific assets' degradation. This contribution is related to *G2.* and includes the papers in Chapters 4, 5, 6, 7 and 8.

C3. A model to evaluate the degree of ownership and contribution alignment.

One of the factors that impacts the degradation of assets is teams' structure and to what extend they own and contribute their code. In order to investigate whether the teams' ownership and contribution alignment impacts the degradation of code, a model was created. The existing metrics to evaluate developer contributions to code are one-dimensional, i.e., they only include one aspect. The presented model in Chapter 6 is created to calculate the degree of ownership and contribution alignment using multiple metrics. The model is designed to adopt based on the availability of the data, i.e., metrics can be added or removed from the model based on data availability. This contribution is related to *G2.* and includes the papers in Chapters 4, 5, 6, 7 and 8.

C4. Empirical evidence supporting and illustrating the impact of ownership and contribution alignment on asset degradation.

The thesis presents two case studies that have used the ownership and contribution alignment model (OCAM) to investigate and illustrate the impact of ownership and contribution alignment on asset degradation. This contribution is related to *G2.* and includes the papers in Chapters 7 and 8.

1.7 Discussion

Assets, other than those which have been extensively examined and studied, such as code, are an integral part of any software development organisations. Assets degrade over time. The continuous maintenance and evolution of software impacts the assets involved in the development process [114, 116].

The recurrent maintenance, and evolution causes assets to degrade. The degradation of assets needs to be managed since they have value for organisations (see Chapter 2).

Asset degradation is unavoidable due to the evolution of the software [114, 116], and all products' assets degrade [106]. Asset degradation comes in different forms: deliberate, unintentional, and entropy. Understanding asset degradation is crucial for managing assets' degradation and mitigating its impact. This is the first step to plan and execute maintenance activities to lessen the impact of asset degradation or possibly avoid the exponential growth of it (see Chapter 2)

One of the aspects that highlights the importance of understanding asset degradation is its eventual financial impact on the development of software-intensive products and services [14, 106]. The degradation of assets leads delays and difficulties in development activities as they increase the cost of change on assets [20]. The financial costs of degradation are a major concern and topic of research in the past years [9, 28, 136, 210]. Therefore, software-developing organisations take significant interest in understanding asset degradation and managing it.

Our research investigating software assets in industry led to the creation of a taxonomy of assets. The taxonomy provides an initial set of assets and asset types (see Chapter 3). The knowledge about degradation and the awareness of organisation's assets can support management activities [178].

Asset management activities increase the cost of software development. Software developing organisations can priorities management activities to target more severe degradation on any particular asset. To achieve this, assets need to be measured and monitored.

The state of assets' degradation can be assessed using different metrics. One of the metrics used to evaluate the degradation of assets is the technical debt (TD), e.g., code technical debt, as measured by industrial tools, to measure the degradation of code. TD has traditionally been used to measure the consequences of short-term decisions on long-term software development [14, 106]. The TD metaphor is widely spread, and has been studied by many researchers [178], and it has turned into a topic of interest for both academia and industry, and it has grown from a metaphor to practice [107]. Therefore, TD provides a good measure to assess the degradation of assets since it is extensively studied in the literature, it is a familiar concept to practitioners, and it has become a practice in industry [106, 178].

The utilisation of methods, metrics, and models to understand the factors impacting asset degradation is crucial. For example, technical debt is one of the common, well-studied, and in-practice metrics used by many companies to survey and investigate particular assets [30, 178]. Technical debt can be measured in different assets, for example, source code or tests. However, understanding the relevant contextual factors that might impact asset's degradation

will facilitate the interpretation of the causes of the degradation. For example, understanding the degree of contribution to source code can help interpret its degradation [210].

Throughout this thesis, we illustrate the impact of asset degradation on developing software-intensive products or services. The results suggest that the degradation can be the consequence of different factors, for example: i) how the accumulation of technical debt associated to different development activities (i.e., to new development, bug fixing, or refactoring) impact code degradation (see Chapter 4); ii) how degradation 'survives' (i.e., the survivability of code smells, bugs, and vulnerabilities as TD items); and how the misalignment between ownership and contribution impacts the faster accumulation of asset degradation (see Chapters 7 and 8). These studies illustrate the complexity of the asset degradation phenomena in software development and highlight the necessity to perform empirical studies to understand individual factors.

The overall takeaway is that understanding asset degradation is the necessary first step to plan sufficient asset management activities [15, 128]. These activities include proactive and reactive solutions to mitigate the impact of asset degradation. In order to achieve this, software developing organisations, together with the research community, need to identify software assets, raise the awareness about asset degradation, investigate the factors that impact the degradation of assets, and be able to manage the degradation.

1.8 Implications for Research and Practice

Here are the main implications for researchers and practitioners based on the findings of this thesis:

- In large organisations, the development of software-intensive products and services is closely intertwined with the social and organisational aspects of work. This is evidenced by the abundance of assets, such as *Business Models* and *Product Management Documentation*, that are linked to the social and organisational aspects of development. Therefore, there is a need to define and establish standards for these assets, including their perceptions and methods for measurement and monitoring.

- Organisation factors seem to have an impact on how assets degrade, therefore software development organisation have to consider these factors (e.g., who are responsible for a particular component) to optimise their asset

management processes and also to raise awareness about the importance of understanding ownership to avoid architectural knowledge vaporisation.

- We have studied a limited subset of assets, and a limited set of factors. Degradation might behave differently in other assets (e.g., in test code), and different roles (e.g., architects or testers) might behave differently when it comes to introduce or mitigate asset degradation, and this needs to be taken into account when planning asset management activities.

1.9 Conclusions and Future Work

This thesis presents the research conducted on software assets, and in particular, asset degradation. Our aspiration is to establish asset degradation in software engineering. We argue that understanding assets and asset degradation is vital to improving software development practices. To reach this goal, we have conducted multiple empirical studies. We included industrial data to strengthen the findings of the thesis. The research collaboration with five companies investigating more than 20 million lines of code has led to the results presented in this thesis.

The contributions of this thesis is two-fold: (i) defining and understanding assets and asset degradation; and (ii) providing evidence on the impact of different factors on the faster accumulation of asset degradation. We are aware that there are a plethora of assets and numerous factors impacting them. Given the wide scope of the topic and the challenges studying them in the limited time, we have focused on a particular, narrow scope, i.e., source code and architecture.

Studying and understanding asset degradation is challenging due to its large scope. There are many factors that impact the degradation of individual assets. And to further complicate the situation, is the impact of the degradation of assets on each other, i.e., the propagation of degradation. Finally, the overall impact of the degradation on reducing the pace of delivery of software, drives the research community and the industry to invest in management activities. The research presented in this thesis is the initial step to further investigation assets and asset degradation.

The long-term objective of that this thesis aspires to achieve is to promote asset management in software engineering and particularly in designing and developing software-intensive products and service. As the next steps, we plan the following:

- Our goal is to explore and identify assets' qualities are measured in industry, what tools they use to measure assets' characteristics and properties. We plan to perform studies to identify the metrics and tools used by the industry to support the taxonomy presented in Chapter 3.
- A natural progression of this work is to explore and investigate the propagation set degradation on dependent assets. As we have discussed in Chapter 2, asset degradation can propagate to other dependant, related assets. It is vital to understand the propagation of asset degradation in order to provide effective solutions and tools to mitigate its impact.
- The research presented in this thesis focuses on certain assets, i.e., source-code- and architecture-related assets. We plan to investigate assets that are less explored and studied such as, assets related to organisation.

References

[1] Al Mamun, M. A., Martini, A., Staron, M., Berger, C., and Hansson, J. Evolution of technical debt: An exploratory study. In *2019 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-Mensura 2019, Haarlem, The Netherlands, October 7-9, 2019* (2019), vol. 2476, CEUR-WS, pp. 87–102.

[2] Alégroth, E., and Gonzalez-Huerta, J. Towards a Mapping of Software Technical Debt onto Testware. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications* (Vienna, Austria, 2017), pp. 404–411.

[3] AlOmar, E. A., Rodriguez, P. T., Bowman, J., Wang, T., Adepoju, B., Lopez, K., Newman, C., Ouni, A., and Mkaouer, M. W. How do developers refactor code to improve code reusability? In *Reuse in Emerging Software Engineering Practices: 19th International Conference on Software and Systems Reuse, ICSR 2020, Hammamet, Tunisia, December 2–4, 2020, Proceedings 19* (2020), Springer, pp. 261–276.

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

[5] Alves, N. S., Ribeiro, L. F., Caires, V., Mendes, T. S., and Spínola, R. O. Towards an ontology of terms on technical debt. In *2014 Sixth International Workshop on Managing Technical Debt* (2014), IEEE, pp. 1–7.

[6] Amadi-Echendu, J. E., Brown, K., Willett, R., and Mathew, J. *Definitions, Concepts and Scope of Engineering Asset Management*, vol. 1. Springer London, London.

[7] Amanatidis, T., Mittas, N., Moschou, A., Chatzigeorgiou, A., Ampatzoglou, A., and Angelis, L. Evaluating the agreement among technical debt measurement tools: building an empirical benchmark of technical debt liabilities. *Empirical Software Engineering 25* (2020), 4161–4204.

[8] Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., and Chatzigeorgiou, A. A Financial Approach for Managing Interest in Technical Debtb. In *International Symposium on Business Modeling and Software Design* (2016), pp. 117–133.

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

[10] Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., Avgeriou, P., and Stamelos, I. Reusability index: A measure for assessing software assets reusability. In *International Conference on Software Reuse* (2018), Springer, pp. 43–58.

[11] Arcoverde, R., Garcia, A., and Figueiredo, E. Understanding the longevity of code smells: preliminary results of an explanatory survey. In *Proceedings of the 4th Workshop on Refactoring Tools* (2011), ACM, pp. 33–36.

[12] Arvanitou, E.-M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., and Stamelos, I. Monitoring technical debt in an industrial setting. In *Proceedings of the Evaluation and Assessment on Software Engineering* (2019), ACM, pp. 123–132.

[13] Arvanitou, E.-M., Nikolaidis, N., Ampatzoglou, A., and Chatzigeorgiou, A. Practitioners' perspective on practices for preventing technical debt accumulation in scientific software development. In *ENASE* (2022), pp. 282–291.

[14] Avgeriou, P., Kruchten, P., Ozkaya, I., and Seaman, C. Managing technical debt in software engineering (dagstuhl seminar 16162). In

Dagstuhl Reports (2016), vol. 6, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[15] Avgeriou, P. C., Taibi, D., Ampatzoglou, A., Fontana, F. A., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, A., Pigazzini, I., et al. An overview and comparison of technical debt measurement tools. *Ieee software 38*, 3 (2020), 61–71.

[16] Badampudi, D., Wohlin, C., and Petersen, K. Experiences from using snowballing and database searches in systematic literature studies. In *Proceedings of the 19th International Conference on Evaluation and Assessment in Software Engineering* (2015), pp. 1–10.

[17] Bao, L., Li, T., Xia, X., Zhu, K., Li, H., and Yang, X. How does working from home affect developer productivity?–a case study of baidu during covid-19 pandemic. *arXiv preprint arXiv:2005.13167* (2020).

[18] Barley, J. M., and Latanfi, B. Bystander intervention in emergencies: Diffusion of responsibility. *Journal of Personality and Social Psychology 8* (1968), 377–383.

[19] Bass, L., Clements, P., and Kazman, R. *Software architecture in practice*. Addison-Wesley Professional, 2003.

[20] Bass, L., Clements, P., and Kazman, R. *Software architecture in practice*. Addison-Wesley Professional, 2021.

[21] Bavani, R. Distributed agile, agile testing, and technical debt. *IEEE software 29*, 6 (2012), 28–33.

[22] Bavota, G., De Carluccio, B., De Lucia, A., Di Penta, M., Oliveto, R., and Strollo, O. When does a refactoring induce bugs? an empirical study. In *2012 IEEE 12th International Working Conference on Source Code Analysis and Manipulation* (2012), IEEE, pp. 104–113.

[23] Bavota, G., De Lucia, A., Di Penta, M., Oliveto, R., and Palomba, F. An experimental investigation on the innate relationship between quality and refactoring. *Journal of Systems and Software 107* (2015), 1–14.

[24] Baškarada, S., Nguyen, V., and Koronios, A. Architecting microservices: Practical opportunities and challenges. *Journal of Computer Information Systems 60* (9 2020), 428–436.

[25] BenIdris, M. Investigate, identify and estimate the technical debt: a systematic mapping study. *International Journal of Software Engineering and Applications (IJSEA) 9*, 5 (2020).

[26] BenIdris, M., Ammar, H., and Dzielski, D. The technical debt density over multiple releases and the refactoring story. *International Journal of Software Engineering and Knowledge Engineering 31*, 01 (2021), 99–116.

[27] Berner, S., Weber, R., and Keller, R. K. Observations and lessons learned from automated testing. In *Proceedings. 27th International Conference on Software Engineering, 2005. ICSE 2005.* (2005), pp. 571–579.

[28] Besker, T., Martini, A., and Bosch, J. The pricey bill of technical debt: When and by whom will it be paid? In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2017), IEEE, pp. 13–23.

[29] Besker, T., Martini, A., and Bosch, J. Time to pay up: Technical debt from a software quality perspective. In *CIbSE* (2017), pp. 235–248.

[30] Besker, T., Martini, A., and Bosch, J. Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software 135* (2018), 1–16.

[31] Besker, T., Martini, A., and Bosch, J. Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work. *Journal of Systems and Software 156* (2019), 41–61.

[32] Bird, C., Nagappan, N., Murphy, B., Gall, H., and Devanbu, P. Don't touch my code! examining the effects of ownership on software quality. In *Proceedings of the 19th ACM SIGSOFT and the 13th European conference on Foundations of software engineering* (2011), pp. 4–14.

[33] Blessing, L. T., and Chakrabarti, A. *DRM: A design reseach methodology.* Springer, 2009.

[34] Blum, B. I. A taxonomy of software development methods. *Communications of the ACM 37*, 11 (1994), 82–94.

[35] Bosch, J. Software architecture: The next step. In *Software Architecture: First European Workshop, EWSA 2004, St Andrews, UK, May 21-22, 2004. Proceedings 1* (2004), Springer, pp. 194–199.

[36] Botsch, B. Significance and measures of association, scopes and methods of political science, 2011.

[37] Bourque, P., Fairley, R. E., et al. *Guide to the software engineering body of knowledge (SWEBOK (R)): Version 3.0*. IEEE Computer Society Press, 2014.

[38] Boylan, J. E., Goodwin, P., Mohammadipour, M., and Syntetos, A. A. Reproducibility in forecasting research. *International Journal of Forecasting 31*, 1 (2015), 79–90.

[39] Branco, A., Cohen, K. B., Vossen, P., Ide, N., and Calzolari, N. Replicability and reproducibility of research results for human language technology: Introducing an lre special section, 2017.

[40] Britto, R., Wohlin, C., and Mendes, E. An extended global software engineering taxonomy. *Journal of Software Engineering Research and Development 4*, 1 (2016), 1–24.

[41] Brooks, F. P. The mythical man-month. *Datamation 20*, 12 (1974), 44–52.

[42] Broughton, V. *Essential classification*. Facet Publishing, 2015.

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

[44] Brown, W. J., Malveau, R. C., Mowbray, T. J., and Wiley, J. *AntiPatterns: Refactoring Software , Architectures, and Projects in Crisis*, vol. 3. Wiley, 1998.

[45] Broy, M. A logical approach to systems engineering artifacts: semantic relationships and dependencies beyond traceability—from requirements to functional and architectural views. *Software & Systems Modeling 17*, 2 (2018), 365–393.

[46] Casale, G., Chesta, C., Deussen, P., Di Nitto, E., Gouvas, P., Koussouris, S., Stankovski, V., Symeonidis, A., Vlassiou, V., Zafeiropoulos, A., et al. Current and future challenges of software engineering for services and applications. *Procedia computer science 97* (2016), 34–42.

[47] Chapin, N., Hale, J. E., Khan, K. M., Ramil, J. F., and Tan, W.-G. Types of software evolution and software maintenance. *Journal of software maintenance and evolution: Research and Practice 13*, 1 (2001), 3–30.

[48] Chatzigeorgiou, A., and Manakos, A. Investigating the evolution of bad smells in object-oriented code. In *2010 Seventh International Conference on the Quality of Information and Communications Technology* (2010), IEEE, pp. 106–115.

[49] Cicchetti, A., Borg, M., Sentilles, S., Wnuk, K., Carlson, J., and Papatheocharous, E. Towards software assets origin selection supported by a knowledge repository. In *2016 1st International Workshop on Decision Making in Software ARCHitecture (MARCH)* (2016), IEEE, pp. 22–29.

[50] Codabux, Z., and Williams, B. Managing technical debt: An industrial case study. In *2013 4th International Workshop on Managing Technical Debt (MTD)* (2013), IEEE, pp. 8–15.

[51] Coghlan, D., and Brannick, T. *Doing Action Research in Your Own Organization (4th ed.)*. London: Sage, 2014.

[52] Constantopoulos, P., and Doerr, M. Component classification in the software information base. *Object-Oriented Software Composition* (1995), 177.

[53] Conway, M. E. How do committees invent. *Datamation 14*, 4 (1968), 28–31.

[54] Creswell, J. W., and Creswell, J. D. *Research design: Qualitative, quantitative, and mixed methods approaches*. Sage publications, 2017.

[55] Cunningham, W. The WyCash portfolio management system. *Proc. Object-Oriented Programming Systems, Languages, and Applications (OOPSLA '92) (Addendum) 4*, 2 (apr 1993), 29–30.

[56] Curtis, B., Sappidi, J., and Szynkarski, A. Estimating the principal of an application's technical debt. *IEEE software 29*, 6 (2012), 34–42.

[57] de Bassi, P. R., Wanderley, G. M. P., Banali, P. H., and Paraiso, E. C. Measuring developers' contribution in source code using quality metrics. In *2018 IEEE 22nd International Conference on Computer Supported Cooperative Work in Design ((CSCWD))* (2018), IEEE, pp. 39–44.

[58] Diamantopoulos, T., Papamichail, M. D., Karanikiotis, T., Chatzidimitriou, K. C., and Symeonidis, A. L. Employing contribution and quality metrics for quantifying the software development process. In *Proceedings of the 17th International Conference on Mining Software Repositories* (2020), pp. 558–562.

[59] Digkas, G., Chatzigeorgiou, A. N., Ampatzoglou, A., and Avgeriou, P. C. Can clean new code reduce technical debt density. *IEEE Transactions on Software Engineering* (2020).

[60] Digkas, G., Lungu, M., Avgeriou, P., Chatzigeorgiou, A., and Ampatzoglou, A. How do developers fix issues and pay back technical debt in the Apache ecosystem? In *IEEE 25th International Conference on Software Analysis, Evolution and Reengineering* (2018), pp. 153–163.

[61] Digkas, G., Lungu, M., Chatzigeorgiou, A., and Avgeriou, P. The evolution of technical debt in the apache ecosystem. European Conference on Software Architecture, pp. 51–66.

[62] Dingsøyr, T., Faegri, T. E., and Itkonen, J. What is large in large-scale? a taxonomy of scale for agile software development. In *International Conference on Product-Focused Software Process Improvement* (2014), vol. 8892, pp. 273–276.

[63] dos Santos, P. S. M., Varella, A., Dantas, C. R., and Borges, D. B. Visualizing and managing technical debt in agile development: An experience report. In *Agile Processes in Software Engineering and Extreme Programming: 14th International Conference, XP 2013, Vienna, Austria, June 3-7, 2013. Proceedings 14* (2013), Springer, pp. 121–134.

[64] Eken, B., Palma, F., Ayşe, B., and Ayşe, T. An empirical study on the effect of community smells on bug prediction. *Software Quality Journal 29* (2021), 159–194.

[65] Ernst, N. A. On the role of requirements in understanding and managing technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)* (2012), pp. 61–64.

[66] Felderer, M., and Travassos, G. H. The evolution of empirical methods in software engineering. In *Contemporary Empirical Methods in Software Engineering*. Springer, 2020, pp. 1–24.

[67] Felizardo, K. R., and Carver, J. C. Automating systematic literature review. *Contemporary Empirical Methods in Software Engineering* (2020), 327–355.

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

[69] Field, A. *Discovering statistics using IBM SPSS statistics*. Sage, 2018.

[70] Flyvbjerg, B. Five misunderstandings about case-study research. *Qualitative Inquiry 12* (2006), 219–245.

[71] Forsgren, N. Octoverse spotlight: An analysis of developer productivity, work cadence, and collaboration in the early days of covid-19, 2020.

[72] Foucault, M., Falleri, J.-R., and Blanc, X. Code ownership in open-source software. In *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering* (2014), pp. 1–9.

[73] Fowler, M. Code ownership, 2006.

[74] Fowler, M. CodeSmell, 2006.

[75] Fowler, M. Microservices, 2014.

[76] Fowler, M. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 2018.

[77] Fowler, M., Beck, K., Brant, J., Opdyke, W., and Roberts, D. *Refactoring: Improving the Design of Existing Code*, 1st ed. Addison Wesley, 1999.

[78] Fox, M., Green, G., and Martin, P. *Doing practitioner research*. Sage, 2007.

[79] Fujiwara, K., Fushida, K., Yoshida, N., and Iida, H. Assessing refactoring instances and the maintainability benefits of them from version archives. In *International Conference on Product Focused Software Process Improvement* (2013), Springer, pp. 313–323.

[80] Garriga, M. Towards a taxonomy of microservices architectures. In *International Conference on Software Engineering and Formal Methods* (2017), Springer, pp. 203–218.

[81] Glass, R. L., and Vessey, I. Contemporary application-domain taxonomies. *IEEE Software 12*, 4 (1995), 63–76.

[82] Glass, R. L., Vessey, I., and Ramesh, V. Research in software engineering: an analysis of the literature. *Information and Software technology 44*, 8 (2002), 491–506.

[83] Gousios, G., Kalliamvakou, E., and Spinellis, D. Measuring developer contribution from software repository data. In *Proceedings of the 2008 international working conference on Mining software repositories* (2008), pp. 129–132.

[84] Greiler, M., Herzig, K., and Czerwonka, J. Code ownership and software quality: A replication study. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories* (2015), IEEE, pp. 2–12.

[85] Griffith, I., Taffahi, H., Izurieta, C., and Claudio, D. A simulation study of practical methods for technical debt management in agile software development. In *Proceedings of the Winter Simulation Conference 2014* (2014), IEEE, pp. 1014–1025.

[86] Grubb, P., and Takang, A. A. *Software maintenance: concepts and practice*. World Scientific, 2003.

[87] Guaman, D., Sarmiento, P., Barba-Guamán, L., Cabrera, P., and Enciso, L. Sonarqube as a tool to identify software metrics and technical debt in the source code through static analysis. In *7th International Workshop on Computer Science and Engineering, WCSE* (2017), pp. 171–175.

[88] Hamill, P. *Unit test frameworks: tools for high-quality software development*. " O'Reilly Media, Inc.", 2004.

[89] Idowu, S., Strüber, D., and Berger, T. Asset management in machine learning: State-of-research and state-of-practice. *ACM Comput. Surv.* (jun 2022).

[90] ISO/IEC. IEC 27005:2018 Information technology–security techniques–information security risk management. Tech. Rep. 0, 2018.

[91] ISO/IEC/IEEE. Systems and software engineering. ISO/IEC/IEEE 24765:2010. Tech. rep., ISO/IEC/IEEE, 2010.

[92] ISO/IEC/IEEE. Asset management - Overview, principles, andsch terminology ISO/IEC/IEEE 55000:2014. Tech. rep., ISO/IEC/IEEE, 2014.

[93] Ivarsson, M., and Gorschek, T. A method for evaluating rigor and industrial relevance of technology evaluations. *Empirical Software Engineering 16*, 3 (2011), 365–395.

[94] Jamshidi, P., Pahl, C., Mendonça, N. C., Lewis, J., and Tilkov, S. Microservices: The journey so far and challenges ahead. *IEEE Software 35* (2018), 24–35.

[95] Kamola, M. How to verify conway's law for open source projects. *IEEE Access 7* (2019), 38469–38480.

[96] Kaplan, E. L., and Meier, P. Nonparametric estimation from incomplete observations. *Journal of the American statistical association 53*, 282 (1958), 457–481.

[97] Kernighan, B. W. 'programming in c- a tutorial. *Unpublished internal memorandum, Bell Laboratories* (1974).

[98] Khurum, M., Gorschek, T., and Wilson, M. The software value map—an exhaustive collection of value aspects for the development of software intensive products. *Journal of software: Evolution and Process 25*, 7 (2013), 711–741.

[99] Kim, M., Zimmermann, T., and Nagappan, N. A field study of refactoring challenges and benefits. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12* (2012), Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering - FSE '12, ACM Press, p. 1.

[100] Kitchenham, B., Brereton, O. P., Budgen, D., Turner, M., Bailey, J., and Linkman, S. Systematic literature reviews in software engineering–a systematic literature review. *Information and software technology 51*, 1 (2009), 7–15.

[101] Klotins, E., and Gorschek, T. Continuous software engineering in the wild. In *Software Quality: The Next Big Thing in Software Engineering and Quality* (Cham, 2022), Springer International Publishing, pp. 3–12.

[102] Klotins, E., Unterkalmsteiner, M., and Gorschek, T. Software-intensive product engineering in start-ups: a taxonomy. *IEEE Software 35*, 4 (2018), 44–52.

[103] Kroll, P., and Kruchten, P. *The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP: A Practitioner's Guide to the RUP*. Addison-Wesley Professional, 2003.

[104] Kruchten, P. The rational unified process 2nd edition: An introduction, 2000.

[105] Kruchten, P. *The rational unified process: an introduction*. Addison-Wesley Professional, 2004.

[106] Kruchten, P., Nord, R., and Ozkaya, I. *Managing Technical Debt: Reducing Friction in Software Development*. Addison-Wesley Professional, 2019.

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

[108] Kruchten, P. B. The 4+ 1 view model of architecture. *IEEE software 12*, 6 (1995), 42–50.

[109] Kujala, S., and Miron-Shatz, T. Emotions, experiences and usability in real-life mobile phone use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2013), pp. 1061–1070.

[110] Kumar, A., Nori, K. V., Natarajan, S., and Lokku, D. S. Value matrix: From value to quality and architecture. In *Economics-Driven Software Architecture*. Elsevier, 2014, pp. 205–240.

[111] Kwasnik, B. H. The role of classification structures in reflecting and building theory. *Advances in Classification Research Online 3*, 1 (1992), 63–82.

[112] Larivière, V., Pontille, D., and Sugimoto, C. R. Investigating the division of scientific labor using the contributor roles taxonomy (credit). *Quantitative Science Studies 2*, 1 (2021), 111–128.

[113] Law, E. L.-C., and Van Schaik, P. Modelling user experience–an agenda for research and practice. *Interacting with computers 22*, 5 (2010), 313–322.

[114] Lehman, M. On understanding laws, evolution, and conservation in the large-program life cycle. *Journal of Systems and Software 1* (1 1979), 213–221.

[115] Lehman, M. M. Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE 68* (1980), 1060–1076.

[116] Lehman, M. M. Laws of software evolution revisited. In *Software Process Technology: 5th European Workshop, EWSPT'96 Nancy, France, October 9–11, 1996 Proceedings 5* (1996), Springer, pp. 108–124.

[117] Lehmann, E. L., and D'Abrera, H. J. *Nonparametrics: statistical methods based on ranks.* Holden-day, 1975.

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

[119] Lenarduzzi, V., and Fucci, D. Towards a holistic definition of requirements debt. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (2019), pp. 1–5.

[120] Lenarduzzi, V., Lomio, F., Huttunen, H., and Taibi, D. Are sonarqube rules inducing bugs? In *2020 IEEE 27th International Conference on Software Analysis, Evolution and Reengineering (SANER)* (2020), IEEE, pp. 501–511.

[121] Lenarduzzi, V., Pecorelli, F., Saarimaki, N., Lujan, S., and Palomba, F. A critical comparison on six static analysis tools: Detection,

agreement, and precision. *Journal of Systems and Software 198* (4 2023), 111575.

[122] Lenarduzzi, V., Saarimaki, N., and Taibi, D. On the diffuseness of code technical debt in java projects of the apache ecosystem. In *2019 IEEE/ACM international conference on technical debt (TechDebt)* (2019), IEEE, pp. 98–107.

[123] Letouzey, J.-L. The sqale method for evaluating technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)* (2012), IEEE, pp. 31–36.

[124] Leung, L. Validity, reliability, and generalizability in qualitative research. *Journal of family medicine and primary care 4*, 3 (2015), 324.

[125] Levén, W., Broman, H., Besker, T., and Torkar, R. The broken windows theory applies to technical debt. *arXiv preprint arXiv:2209.01549* (2022).

[126] Leys, C., Ley, C., Klein, O., Bernard, P., and Licata, L. Detecting outliers: Do not use standard deviation around the mean, use absolute deviation around the median. *Journal of Experimental Social Psychology 49*, 4 (2013), 764–766.

[127] Li, B., Vendome, C., Linares-Vásquez, M., Poshyvanyk, D., and Kraft, N. A. Automatically documenting unit test cases. In *2016 IEEE international conference on software testing, verification and validation (ICST)* (2016), IEEE, pp. 341–352.

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

[129] Lientz, B. P., and Swanson, E. B. *Software maintenance management: A study of the maintenance of computer application software in 487 data processing organizations.* Addison-Wesley, 1980.

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

[131] Lozano, A., Wermelinger, M., and Nuseibeh, B. Assessing the impact of bad smells using historical information. In *Ninth international workshop on Principles of software evolution: in conjunction with the 6th ESEC/FSE joint meeting* (2007), ACM, pp. 31–34.

[132] Mann, H. B., and Whitney, D. R. On a test of whether one of two random variables is stochastically larger than the other. *The annals of mathematical statistics* (1947), 50–60.

[133] Marcilio, D., Bonifácio, R., Monteiro, E., Canedo, E., Luz, W., and Pinto, G. Are static analysis violations really fixed? a closer look at realistic usage of sonarqube. In *2019 IEEE/ACM 27th International Conference on Program Comprehension (ICPC)* (2019), IEEE, pp. 209–219.

[134] Martin, R. C. *Clean Code*. Prentice Hall, 2008.

[135] Martini, A., Besker, T., and Bosch, J. Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. *Science of Computer Programming 163* (2018), 42–61.

[136] Martini, A., and Bosch, J. The magnificent seven: towards a systematic estimation of technical debt interest. In *Proceedings of the XP2017 Scientific Workshops* (2017), pp. 1–5.

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

[138] Martini, A., Stray, V., and Moe, N. B. Technical-, social- and process debt in large-scale agile: An exploratory case-study. In *Agile Processes in Software Engineering and Extreme Programming – Workshops* (Cham, 2019), R. Hoda, Ed., Springer International Publishing, pp. 112–119.

[139] Mauerer, W., Joblin, M., Tamburri, D. A., Paradis, C., Kazman, R., and Apel, S. In search of socio-technical congruence: A large-scale longitudinal study. *arXiv preprint arXiv:2105.08198* (2021).

[140] McCabe, T. J. A complexity measure. *IEEE Transactions on software Engineering*, 4 (1976), 308–320.

[141] Méndez, D., Böhm, W., Vogelsang, A., Mund, J., Broy, M., Kuhrmann, M., and Weyer, T. Artefacts in software engineering: a fundamental positioning. *Software & Systems Modeling 18*, 5 (2019), 2777–2786.

[142] Méndez, D., Penzenstadler, B., Kuhrmann, M., and Broy, M. A meta model for artefact-orientation: fundamentals and lessons learned in requirements engineering. In *International Conference on Model Driven Engineering Languages and Systems* (2010), Springer, pp. 183–197.

[143] Méndez Fernández, D., Böhm, W., Vogelsang, A., Mund, J., Broy, M., Kuhrmann, M., and Weyer, T. Artefacts in software engineering: a fundamental positioning. *Software and Systems Modeling 18*, 5 (oct 2019), 2777–2786.

[144] Meneely, A., and Williams, L. Secure open source collaboration: An empirical study of linus' law. In *16th ACM Conference on Computer and Communications Security* (2009), Association for Computing Machinery, pp. 453–462.

[145] Mens, T., and Demeyer, S. *Software Evolution*. Springer Berlin Heidelberg, 2008.

[146] Menzies, T., Greenwald, J., and Frank, A. Data mining static code attributes to learn defect predictors. *IEEE transactions on software engineering 33*, 1 (2006), 2–13.

[147] Merriam, S. B., and Tisdell, E. J. *Qualitative research: A guide to design and implementation*. John Wiley & Sons, 2015.

[148] Miles, M. B., Huberman, A. M., and Saldaña, J. Qualitative data analysis: A methods sourcebook. 3rd, 2014.

[149] Miller Jr, R. G. *Survival analysis*, vol. 66. John Wiley & Sons, 2011.

[150] Munson, J. C., and Elbaum, S. G. Code churn: A measure for estimating the impact of code change. In *Proceedings. International Conference on Software Maintenance (Cat. No. 98CB36272)* (1998), IEEE, pp. 24–31.

[151] Müter, L., Deoskar, T., Mathijssen, M., Brinkkemper, S., and Dalpiaz, F. Refinement of user stories into backlog items: Linguistic structure and action verbs. In *International Working Conference*

on Requirements Engineering: Foundation for Software Quality (2019), Springer, pp. 109–116.

[152] Nagappan, M., Zimmermann, T., and Bird, C. Diversity in software engineering research. Proceedings of the 2013 9th joint meeting on foundations of software engineering, pp. 466–476.

[153] Nagappan, N., Murphy, B., and Basili, V. The influence of organizational structure on software quality: an empirical case study. In *Proceedings of the 30th international conference on Software engineering* (2008), pp. 521–530.

[154] Neamtiu, I., Xie, G., and Chen, J. Towards a better understanding of software evolution: an empirical study on open-source software. *Journal of Software: Evolution and Process 25*, 3 (2013), 193–218.

[155] Newman, S. *Building Microservices*. O'Reilly, 2015.

[156] Newman, S. *Building microservices*. O'Reilly Media, Inc., 2021.

[157] NIST. NIST Special Publication 800-30 Revision 1 - Guide for Conducting Risk Assessments. Tech. Rep. September, 2012.

[158] Nordbarg, M. E. Managing code ownership. *IEEE Software 20* (3 2003), 26–33.

[159] Northrop, L., Clements, P., Bachmann, F., Bergey, J., Chastek, G., Cohen, S., Donohoe, P., Jones, L., Krut, R., Little, R., et al. A framework for software product line practice, version 5.0. *SEI.–2007–http://www. sei. cmu. edu/productlines/index. html* (2007).

[160] Northrop, L., Feiler, P., Gabriel, R. P., Linger, R., Longstaff, T., Kazman, R., Klein, M., Shcmidt, D., Sullivan, K., and Wallnau, K. Ultra-large-scale systems the software challenge of the future ultra-large-scale systems: The software challenge of the future. Tech. rep., Software Engineering Institute, Carnegie Mellon University, 2006.

[161] Oliveira, E., Fernandes, E., Steinmacher, I., Cristo, M., Conte, T., and Garcia, A. Code and commit metrics of developer productivity: a study on team leaders perceptions. *Empirical Software Engineering 25*, 4 (2020), 2519–2549.

[162] Oliveira, F., Goldman, A., and Santos, V. Managing technical debt in software projects using scrum: An action research. In *2015 Agile Conference* (2015), IEEE, pp. 50–59.

[163] Omair, A., et al. Sample size estimation and sampling techniques for selecting a representative sample. *Journal of Health Specialties 2*, 4 (2014), 142.

[164] Palomba, F., Bavota, G., Penta, M. D., Fasano, F., Oliveto, R., and Lucia, A. D. On the diffuseness and the impact on maintainability of code smells: a large scale empirical investigation. *Empirical Software Engineering 23*, 3 (2018), 1188–1221.

[165] Palomba, F., Bavota, G., Penta, M. D., Oliveto, R., and Lucia, A. D. Do They Really Smell Bad? A Study on Developers' Perception of Bad Code Smells. In *2014 IEEE International Conference on Software Maintenance and Evolution* (sep 2014), IEEE, pp. 101–110.

[166] Palomba, F., Zaidman, A., Oliveto, R., and De Lucia, A. An Exploratory Study on the Relationship between Changes and Refactoring. In *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)* (may 2017), IEEE, pp. 176–185.

[167] Pantiuchina, J., Lanza, M., and Bavota, G. Improving code: The (mis) perception of quality metrics. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)* (2018), IEEE, pp. 80–91.

[168] Parizi, R. M., Spoletini, P., and Singh, A. Measuring team members' contributions in software engineering projects using git-driven technology. In *2018 IEEE Frontiers in Education Conference (FIE)* (2018), IEEE, pp. 1–5.

[169] Peters, R., and Zaidman, A. Evaluating the lifespan of code smells using software repository mining. In *2012 16th European Conference on Software Maintenance and Reengineering* (2012), IEEE, pp. 411–416.

[170] Posnett, D., D'Souza, R., Devanbu, P., and Filkov, V. Dual ecological measures of focus in software development. In *2013 35th International Conference on Software Engineering (ICSE)* (2013), IEEE, pp. 452–461.

[171] Power, K. Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options. In *2013 4th International Workshop on Managing Technical Debt (MTD)* (2013), IEEE, pp. 28–31.

[172] Rahman, F., and Devanbu, P. Ownership, experience and defects: a fine-grained study of authorship. In *Proceedings of the 33rd International Conference on Software Engineering* (2011), pp. 491–500.

[173] Ralph, P., Baltes, S., Adisaputri, G., Torkar, R., Kovalenko, V., Kalinowski, M., Novielli, N., Yoo, S., Devroey, X., Tan, X., et al. Pandemic programming: how covid-19 affects software developers and how their organizations can help. *arXiv preprint arXiv:2005.01127* (2020).

[174] Ralph, P., and Tempero, E. Construct validity in software engineering research and software metrics. In *22nd International Conference on Evaluation and Assessment in Software Engineering* (Christchurch, New Zealand, 2018), Association for Computing Machinery (ACM).

[175] Rapu, D., Ducasse, S., Gîrba, T., and Marinescu, R. Using history information to improve design flaws detection. In *Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.* (2004), IEEE, pp. 223–232.

[176] Raymond, E. The cathedral and the bazaar. *Knowledge, Technology & Policy 12* (1999), 23–49.

[177] Reussner, R., Goedicke, M., Hasselbring, W., Vogel-Heuser, B., Keim, J., and Märtin, L. *Managed software evolution*. Springer Nature, 2019.

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

[179] Robson, C. *Real world research*, vol. 3. Wiley Chichester, 2011.

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

[181] Runeson, P., Host, M., Rainer, A., and Regnell, B. *Case Study Research in Software Engineering: Guidelines and Examples*. John Wiley and Sons, 2012.

[182] Saarimäki, N., Lenarduzzi, V., and Taibi, D. On the diffuseness of code technical debt in java projects of the apache ecosystem. In *Proceedings of the Second International Conference on Technical Debt* (2019), IEEE Press, pp. 98–107.

[183] Saher, N., Baharom, F., and Ghazali, O. Requirement change taxonomy and categorization in agile software development. In *2017 6th International Conference on Electrical Engineering and Informatics (ICEEI)* (2017), IEEE, pp. 1–6.

[184] Saldaña, J. *The coding manual for qualitative researchers*. Sage, 2015.

[185] Santos, J. A. M., Rocha-Junior, J. B., Prates, L. C. L., do Nascimento, R. S., Freitas, M. F., and de Mendonça, M. G. A systematic review on the code smell effect. *Journal of Systems and Software 144*, July (oct 2018), 450–477.

[186] Schneider, J., Gaul, A. J., Neumann, C., Hogräfer, J., Wellssow, W., Schwan, M., and Schnettler, A. Asset management techniques. *International Journal of Electrical Power & Energy Systems 28*, 9 (2006), 643–654.

[187] Schweik, C. M., English, R. C., Kitsing, M., and Haire, S. Brooks' versus linus' law: An empirical test of open source projects. In *9th Annual International Digital Government Research Conference* (2008), pp. 423–424.

[188] Siavvas, M., Tsoukalas, D., Jankovic, M., Kehagias, D., and Tzovaras, D. Technical debt as an indicator of software security risk: a machine learning approach for software development enterprises. *Enterprise Information Systems 16*, 5 (2022), 1824017.

[189] Silva, D., Tsantalis, N., and Valente, M. T. Why we refactor? confessions of GitHub contributors. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering - FSE 2016* (New York, New York, USA, 2016), ACM Press, pp. 858–870.

[190] Silva, M., Oliveira, T., and Bastos, R. Software artifact metamodel. In *2009 XXIII Brazilian Symposium on Software Engineering* (2009), pp. 176–186.

[191] Sjøberg, D. I., and Bergersen, G. R. Construct validity in software engineering. *IEEE Transactions on Software Engineering* (2022), 1–1.

[192] Sjøberg, D. I., Yamashita, A., Anda, B. C., Mockus, A., and Dybå, T. Quantifying the effect of code smells on maintenance effort. *IEEE Transactions on Software Engineering 39*, 8 (2012), 1144–1156.

[193] Šmite, D., Brede Moe, N., Klotins, E., and Gonzalez-Huerta, J. From forced working-from-home to voluntary working-from-anywhere: Two revolutions in telework. *Journal of Systems and Software 195* (2023), 111509.

[194] Šmite, D., Wohlin, C., Galviņa, Z., and Prikladnicki, R. An empirically based terminology and taxonomy for global software engineering. *Empirical Software Engineering 19*, 1 (2014), 105–153.

[195] Sommerville, I. Software engineering. 10th. In *Book Software Engineering. 10th, Series Software Engineering*. Addison-Wesley, 2015.

[196] Stol, K.-J., and Fitzgerald, B. The abc of software engineering research. *ACM Transactions on Software Engineering and Methodology (TOSEM) 27*, 3 (2018), 1–51.

[197] Stringer, E. T. *Action Research (4th Edition)*. Thousand Oaks, CA: Sage, 2014.

[198] Sundelin, A., Gonzalez-Huerta, J., and Wnuk, K. The hidden cost of backward compatibility: When deprecation turns into technical debt - An experience report. In *Proceedings - 2020 IEEE/ACM International Conference on Technical Debt, TechDebt 2020* (2020), pp. 67–76.

[199] Sundelin, A., Gonzalez-Huerta, J., Wnuk, K., and Gorschek, T. Towards an anatomy of software craftsmanship. *ACM Transactions on Software Engineering and Methodology (TOSEM) 31*, 1 (2021), 1–49.

[200] Suryanarayana, G., Samarthyam, G., and Sharma, T. *Refactoring for Software Design Smells: Managing Technical Debt*, vol. 11. Elsevier Science, 2014.

[201] Svahnberg, M., Van Gurp, J., and Bosch, J. A taxonomy of variability realization techniques. *Software: Practice and experience 35*, 8 (2005), 705–754.

[202] Taivalsaari, A., and Mikkonen, T. A taxonomy of iot client architectures. *IEEE software 35*, 3 (2018), 83–88.

[203] Thongtanunam, P., McIntosh, S., Hassan, A. E., and Iida, H. Revisiting code ownership and its relationship with software quality in the scope of modern code review. In *Proceedings of the 38th international conference on software engineering* (2016), pp. 1039–1050.

[204] Tilley, S., and Huang, S. Documenting software systems with views iii: towards a task-oriented classification of program visualization techniques. In *Proceedings of the 20th annual international conference on Computer documentation* (2002), pp. 226–233.

[205] Tofan, D., Galster, M., and Avgeriou, P. Reducing architectural knowledge vaporization by applying the repertory grid technique. In *Software Architecture: 5th European Conference, ECSA 2011, Essen, Germany, September 13-16, 2011. Proceedings 5* (2011), Springer, pp. 244–251.

[206] Tom, E., Aurum, A., and Vidgen, R. A consolidated understanding of technical debt. In *ECIS 2012 Proceedings* (2012).

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

[208] Tornhill, A. *Your Code As A Crime Scene*. The Pragmatic Bookshelf, 2015.

[209] Tornhill, A. *Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis*. Pragmatic Bookshelf, 2018.

[210] Tornhill, A., and Borg, M. Code red: the business impact of code quality-a quantitative study of 39 proprietary production codebases. In *Proceedings of the International Conference on Technical Debt* (2022), pp. 11–20.

[211] Tsantalis, N., Mansouri, M., Eshkevari, L. M., Mazinanian, D., and Dig, D. Accurate and efficient refactoring detection in commit history. In *Proceedings of the 40th International Conference on SoftwarTe Engineering* (2018), ACM, pp. 483–494.

[212] Tsoukalas, D., Kehagias, D., Siavvas, M., and Chatzigeorgiou, A. Technical debt forecasting: an empirical study on open-source repositories. *Journal of Systems and Software 170* (2020), 110777.

[213] Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., and Poshyvanyk, D. When and why your code starts to smell bad. In *Proceedings of the 37th International Conference on Software Engineering-Volume 1* (2015), IEEE Press, pp. 403–414.

[214] Tufano, M., Palomba, F., Bavota, G., Oliveto, R., Di Penta, M., De Lucia, A., and Poshyvanyk, D. When and why your code starts to smell bad (and whether the smells go away). *IEEE Transactions on Software Engineering 43*, 11 (2017), 1063–1088.

[215] Unterkalmsteiner, M., Feldt, R., and Gorschek, T. A taxonomy for requirements engineering and software test alignment. *ACM Transactions on Software Engineering and Methodology (TOSEM) 23*, 2 (2014), 1–38.

[216] Usman, M., Britto, R., Börstler, J., and Mendes, E. Taxonomies in software engineering: A systematic mapping study and a revised taxonomy development method. *Information and Software Technology 85* (2017), 43–59.

[217] Vassallo, C., Panichella, S., Palomba, F., Proksch, S., Zaidman, A., and Gall, H. C. Context is king: The developer perspective on the usage of static analysis tools. In *25th IEEE International Conference on Software Analysis, Evolution and Reengineering* (2018).

[218] Vetro, A. Using automatic static analysis to identify technical debt. In *2012 34th International Conference on Software Engineering (ICSE)* (2012), IEEE, pp. 1613–1615.

[219] von Linné, C. *Systema naturae; sive, Regna tria naturae: systematice proposita per classes, ordines, genera & species.* Haak, 1735.

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

[221] Wohlin, C., Wnuk, K., Šmite, D., Franke, U., Badampudi, D., and Cicchetti, A. Supporting strategic decision-making for selection of software assets. In *International conference of software business* (2016), Springer, pp. 1–15.

[222] Wolfram, K., Martine, M., and Sudnik, P. Recognising the types of software assets and its impact on asset reuse. In *European Conference on Software Process Improvement* (2020), Springer, pp. 162–174.

[223] Yamashita, A., and Moonen, L. Do code smells reflect important maintainability aspects? In *2012 28th IEEE international conference on software maintenance (ICSM)* (2012), IEEE, pp. 306–315.

[224] Yamashita, A., and Moonen, L. Exploring the impact of inter-smell relations on software maintainability: An empirical study. In *Proceedings of the 2013 International Conference on Software Engineering* (2013), IEEE Press, pp. 682–691.

[225] Yin, R. K. *Case study research and applications: Design and Methods*, 6 ed. Sage, 2018.

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

[227] Yoshida, N., Saika, T., Choi, E., Ouni, A., and Inoue, K. Revisiting the relationship between code smells and refactoring. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)* (may 2016), vol. 2016-July, IEEE, pp. 1–4.

[228] Zabardast, E., Frattini, J., Gonzalez-Huerta, J., Mendez, D., Gorschek, T., and Wnuk, K. Assets in software engineering: What are they after all? *Journal of Systems and Software 193* (2022), 111485.

[229] Zabardast, E., Gonzalez-Huerta, J., and Palma, F. The impact of forced working-from-home on code technical debt: An industrial case study. In *2022 48th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2022), IEEE, pp. 298–305.

[230] Zabardast, E., Gonzalez-Huerta, J., and Šmite, D. Refactoring, bug fixing, and new development effect on technical debt: An industrial case study. *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)* (2020), 376–384.

[231] Zabardast, E., Gonzalez-Huerta, J., and Tanveer, B. Ownership vs contribution: Investigating the alignment between ownership and contribution. In *19th IEEE International Conference on Software Architectures* (2022), IEEE.

[232] Zhao, Y., Dong, J., and Peng, T. Ontology classification for semantic-web-based software engineering. *IEEE Transactions on Services Computing 2*, 4 (2009), 303–317.