University of Groningen

Economics-driven Technical Debt Management

Ampatzoglou, Areti

DOI:
[10.33612/diss.157426352](https://doi.org/10.33612/diss.157426352)

IMPORTANT NOTE: You are advised to consult the publisher's version (publisher's PDF) if you wish to cite from it. Please check the document version below.

Document Version
Publisher's PDF, also known as Version of record

Publication date:
2021

[Link to publication in University of Groningen/UMCG research database](#)

Citation for published version (APA):
Ampatzoglou, A. (2021). *Economics-driven Technical Debt Management*. [Thesis fully internal (DIV), University of Groningen]. University of Groningen. <https://doi.org/10.33612/diss.157426352>

Copyright

Other than for strictly personal use, it is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license (like Creative Commons).

The publication may also be distributed here under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license. More information can be found on the University of Groningen website: <https://www.rug.nl/library/open-access/self-archiving-pure/taverne-amendment>.

Take-down policy

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

Downloaded from the University of Groningen/UMCG research database (Pure): <http://www.rug.nl/research/portal>. For technical reasons the number of authors shown on this cover page is limited to 10 maximum.

university of
groningen

Economics-driven Technical Debt Management

PhD thesis

to obtain the degree of PhD at the
University of Groningen
on the authority of the
Rector Magnificus Prof. C. Wijmenga
and in accordance with
the decision by the College of Deans.

This thesis will be defended in public on
Monday 15 February 2021 at 16.15 hours

by

Areti Ampatzoglou

born on 20 October 1976
in Kalamaria, Greece

Supervisor

Prof. dr. P. Avgeriou

Co-supervisor

Dr. A. N. Chatzigeorgiou

Assessment Committee

Prof. C. Seaman

Prof. A. Serebrenik

Prof. A. Capiluppi

Samenvatting

Technical Debt (TD) is een metafoor in software engineering die een analogie maakt tussen kortere wegen in ontwikkeling en het aangaan van een lening. Volgens de metafoor kan een softwareteam of -bedrijf (al of niet opzettelijk) de ontwikkelingsinspanningen vooral reduceren door het nemen van dergelijke kortere wegen, en zo een bepaald geldbedrag uitsparen (leningbedrag–TD Hoofdsom). Aan dit voordeel zijn bijbehorende kosten verbonden in de zin dat het product wordt vrijgegeven in sub-optimale kwaliteit, wat extra onderhoudskosten met zich meebrengt (TD Rente), e.g. extra inspanning voor het oplossen van fouten, begrijpen van de bestaande code, toevoegen van nieuwe functies etc.

Het proces dat de TD van een softwaresysteem in zijn ontwikkeling beheert wordt Technical Debt Management (TDM) genoemd. Waar TD, als metafoor, economische concepten toepast op het gebied van software engineering, heeft de huidige TDM aanpak zich voornamelijk gericht op software-aspecten. Daarentegen is geen rekening gehouden met de financiële aspecten van de metafoor. Dit wordt met name zichtbaar in: (a) het ontbreken van een aanpak en tools voor het genereren van inkomsten uit TD-rente, de feitelijke definitie van TD-rente volgend; (b) geen link kunnen leggen tussen TD-rente en hoofdsom; en (c) het ontbreken van een aanpak die rekening houdt met kosten en financiële voordelen buiten de TD-hoofdsom en rente bij de besluitvorming inzake TDM. Om bovenvermelde beperkingen aan te pakken hebben we, als onderdeel van dit PhD-project, methodes en tools voorgesteld die afkomstig zijn uit de discipline van de economische wetenschap om TD-rente te berekenen, de verhouding tussen TD-rente en hoofdsom te onderzoeken en TDM als een financiële investering te behandelen. Het PhD-project is in vier hoofdstappen georganiseerd, zoals beschreven in de volgende paragrafen.

Als eerste stap bekeken we de status van onderzoek en praktijk inzake TDM aan de hand van twee onderzoeken, die respectievelijk de nadruk leggen op: (a) het corpusonderzoek dat gebruik maakt van financiële theorieën en termen door het uitvoeren van een systematisch literatuuronderzoek; en (b) de perceptie van of belanghebbenden in het bedrijfsleven aangaande TDM door het uitvoeren van een industriële casestudy. De voornaamste bevindingen van het systematische literatuuronderzoek benadrukken dat er te weinig gebruik wordt gemaakt van de discipline van economische wetenschap in TD-onderzoek. Op basis hiervan hebben we een lijst van financiële begrippen in TD voorgesteld, die we visualiseren ten behoeve van begrijpelijkheid. Daarnaast is het aantal

onderzoeken die TD-rente verkennen lager gebleken dan die welke gaan over TD-hoofdsom. Anderzijds bleek uit de bevindingen van het onderzoek in het bedrijfsleven dat TDM cruciaal is voor alle toepassingsgebieden, maar alleen is beperkt tot projecten die naar verwachting een lange levensduur hebben, met intensieve onderhoudsactiviteiten. Door de resultaten van beide onderzoeken te overwegen kunnen we stellen dat code en architectuur van TD de meest prominente soorten van TD zijn, en dat de onderhoudsperiode een belangrijke parameter is die moet worden meegenomen bij het uitvoeren van TDM.

De tweede stap is gericht op de eerste beperking (i.e., de berekening van TD-rente). Voor het bereiken van kwantificatie van TD-rente hebben we als input de twee belangrijkste resultaten van de vorige stap genomen: we hebben ons op TD-code gericht, en we hebben rekening gehouden met de tijdsduur van softwareonderhoud (we merken op dat we ons richten op architectural debt in de eerste stap van dit PhD-project). We hebben met name eerst theoretische concepten van rente uit de economie aangepast aan het TD-domein, en hebben het Framework for managing Interest in Technical Debt (FITTED) geïntroduceerd. Als onderdeel van FITTED introduceerden we het Breekpunt-concept, dat overeenkomt met de periode waarin het bestaan van TD financieel niet nadelig is voor de evolutie van het softwaresysteem. Als laatste stap stelden we een aanpak voor kwantificatie van TD-rente voor op basis van bekende voorspellende factoren inzake onderhoudbaarheid en historische gegevens. De validatie door het bedrijfsleven van de voorgestelde aanpak toonde aan dat onze berekening sterk is gecorreleerd met de perceptie van aandeelhouders, met betrekking tot onderhoudsboetes (TD-rente) en duurzaamheid van onderhoud (Breekpunt). Alle bovenvermelde benaderingen zijn geïmplementeerd in een tool, om toepasbaarheid te vergroten en schaalbaarheid te verbeteren.

In de derde stap pakken we de tweede beperking aan door empirisch de bouwsteen van rente-theorie inzake financiën te onderzoeken: het bestaan van een rentepercentage bij TD. In financiën is het rentepercentage de parameter waarmee rente wordt berekend op basis van de hoofdsom. Echter, rentepercentage is niet van toepassing bij TD, aangezien TD-rente niet kan worden berekend als een deel van de TD-hoofdsom. Desalniettemin zou zelfs een niet-lineaire relatie tussen de twee concepten de metafoor versterken en ook onze theorie over TD-rente bevestigen. Hiervoor hebben we empirisch de relatie onderzocht tussen hoofdsom en rente van TD op broncodeniveau, en hebben we het eerste gedegen empirisch bewijs geleverd dat een dergelijke relatie kan worden gecreëerd: groepen met soortgelijke niveaus van TD-hoofdsom genereren meestal een soortgelijk TD-renteniveau.

Als een vierde en laatste stap richtten we ons op de derde vastgestelde beperking en stelden een aanpak en een bijbehorende tool voor die kosten en financiële voordelen in de besluitvorming inzake TDM overwegen. De reden hiervoor was de noodzaak om rekening rekening te houden met verscheidene technische en financiële parameters tijdens de besluitvorming wanneer TD moest worden terugbetaald. Om dit doel te bereiken stellen we een architectureel besluitvormingsproces voor, genaamd Architectural Decision-Making as a Financial Investment (ADMIT). In ADMIT worden alle parameters van een besluit getransformeerd tot kosten en baten, en worden ze vervolgens beoordeeld middels een traditionele kosten-baten analyse. De industriële validatie van ADMIT suggereerde dat de voorgestelde aanpak eenvoudig is, en het belangrijkste voordeel ervan is juist dat de kwantificatie van alle parameters in geld wordt uitgedrukt, zodat communicatie met management wordt vergemakkelijkt en veelsoortige input kan worden samengevoegd.

Technical Debt (TD) is a software engineering metaphor that draws an analogy between shortcuts in development and taking out a loan. In particular, the metaphor considers that a software team or company (intentionally or unintentionally) can reduce the development effort by taking such shortcuts, and thus save a specific amount of money (amount of loan—***TD Principal***). This benefit comes with an associated cost, in the sense that the product is released with sub-optimal quality, leading to the occurrence of extra maintenance costs (***TD Interest***), e.g., additional effort for bug fixing, understanding the existing code, adding new features.

The process that deals with the TD of a software system along its evolution, is termed ***Technical Debt Management*** (TDM). While TD, as a metaphor, applies economics concepts to the field of software engineering, current TDM approaches have focused predominantly on software aspects. In contrast, the financial aspects of the metaphor have not been taken into account. This is particularly manifested through: (a) a lack of approaches and tools for monetizing TD Interest, following the actual definition of TD Interest; (b) a lack of establishing a relation between TD Interest and Principal; and (c) a lack of approaches that consider costs and financial benefits beyond TD Principal and Interest in the decision-making of TDM. To tackle the aforementioned limitations, as part of this PhD project, we have ***proposed approaches and tools that stem from the discipline of economics*** to quantify TD Interest, explore the relation between TD Interest and Principal and treat TDM as a financial investment. The PhD project was organized into four main steps, as described in the following paragraphs.

As a first step, we ***reviewed the state-of-research and -practice*** on TDM, through two studies, respectively placing emphasis on: (a) the corpus of research that uses financial theories and terms—by conducting a systematic literature review; and (b) the perception of industrial stakeholders regarding TDM by conducting an industrial case study. The main findings of the systematic literature review highlight the under-utilization of the economics discipline in TD research. Based on this, we have proposed a glossary of financial terms in TD, which we visualize for understandability purposes. Additionally, the amount of studies that explore TD Interest has proven to be lower than those that treat TD Principal. On the other hand, the findings of the industrial survey suggested that TDM is crucial for all application domains, but limited only to projects that have an expected long lifetime, with intense maintenance activities. By considering the results of both studies, we can argue that

code and architecture TD are the most prominent types of TD, and that the maintenance period is an important parameter that must be considered when performing TDM.

The second step, focuses on the first limitation (i.e., the calculation of TD Interest). To achieve ***TD Interest quantification***, we have taken as input the two main outcomes of the previous step: we focused on code TD, and we considered the time period of software maintenance (we note that we focus on architecture debt in the first step of this PhD project). In particular, we first adapted interest theory concepts from economics to fit the TD domain, introducing the Framework for managing Interest in Technical Debt (FITTED). As part of FITTED, we introduced the Breaking Point concept, which corresponds to the time period for which the existence of TD is not financially harmful for the evolution of the software system. As a final step, we proposed an approach for quantifying TD Interest, based on well-known maintainability predictors and historical data. The industrial validation of the proposed approach revealed that our calculation is strongly correlated to the perception of stakeholders, with respect to maintenance penalties (TD Interest) and maintenance sustainability (Breaking Point). All the aforementioned approaches have been implemented in a tool, so as to boost applicability and improve scalability.

In the third step, we tackle the second limitation by empirically exploring the cornerstone of interest theory in finance: the ***existence of an interest rate in TD***. In finance, the interest rate is the parameter through which Interest is calculated, based on Principal. However, interest rate is not applicable in TD, since TD Interest cannot be calculated as a fraction of TD Principal. Nevertheless, even a non-linear relation between the two concepts would strengthen the metaphor and also confirm our TD interest theory. To this end, we have empirically explored the relation between TD Principal and TD Interest at the source code level and provided the first solid empirical evidence that such a relation can be formed: classes with similar levels of TD Principal tend to generate a similar level of TD Interest.

As a fourth and final step, we focused on the third identified limitation and proposed an approach and an accompanying tool that considers **costs and financial benefits in the decision making of TDM**. This was motivated by the need to take several technical and financial parameters into account during decision-making on when to repay TD. To achieve this goal, we propose an architectural decision-making approach, termed Architectural Decision-Making as a Financial Investment (ADMIT). In ADMIT, all parameters of a decision are transformed to costs and benefits, and are subsequently assessed using a traditional cost-benefit analysis. The industrial validation of ADMIT suggested that the proposed approach is straightforward, and its main benefit is exactly the quantification of all parame-

ters in a monetized form, so as to ease communication with management and enable the aggregation of heterogeneous inputs.

Table of Contents

Samenvatting.....	I
Abstract.....	IV
Table of Contents	VII
Acknowledgements	XI
Chapter 1 - Introduction.....	13
1.1 Technical Debt Concepts.....	13
1.2 Technical Debt Management.....	15
1.3 Research Design.....	16
Chapter 2 – The Financial Aspect of Managing Technical Debt: A Systematic Literature Review	25
2.1 Motivation	25
2.2 Related Work.....	27
2.3 Background Information	29
2.4 Review Methodology	32
2.5 Results	37
2.6 Discussion.....	47
2.7 Threats to Validity	57
2.8 Conclusion	58
Chapter 3 – The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study	60
3.1 Motivation	60
3.2 Related Work.....	62
3.3 Empirical Methodology.....	63
3.4 Results	69
3.5. Implications to researchers and practitioners	74

3.6 Threats to Validity.....	75
3.7 Conclusion	76
Chapter 4 – A Financial Approach for Managing Interest in Technical Debt	77
4.1 Motivation	77
4.2 Background Information	78
4.3 Empirical Methodology.....	81
4.4 Framework for Managing Interest in TD	83
4.5 Research Implications	87
4.6 Illustrative Example.....	88
4.7 Threats to Validity	90
4.8 Conclusion	90
Chapter 5 – A Framework for Managing Interest in Technical Debt: An Industrial Validation.....	91
5.1 Motivation	91
5.2 FITTED Interest Theory	92
5.3 FITTED Instantiation	94
5.4 Empirical Methodology.....	100
5.5 Results	101
5.6 Discussion.....	104
5.7 Threats to Validity	105
5.8 Conclusion	106
Chapter 6 – Exploring the Relation between Technical Debt Principal and Interest: An Empirical Approach.....	107
6.1 Motivation	107
6.2 Related Work.....	109
6.3 Background Information	115
6.4 Empirical Methodology	120

6.5 Results	128
6.6 Discussion.....	134
6.7 Threats to Validity	137
6.8 Conclusion	139
Chapter 7 – Architectural Decision-making as a Financial Investment: An Industrial Case Study.....	141
7.1 Motivation	141
7.2 Related Work.....	144
7.3 Background Information	146
7.4 Architectural Decision-making as a Financial Investment	151
7.5 Empirical Methodology.....	157
7.6 Results	162
7.7 Discussion.....	174
7.8 Threats to Validity	177
7.9 Conclusion	178
Chapter 8 – Conclusions and Future Work.....	179
8.1 Answers to Research Questions and Contributions.....	179
8.2 Ongoing and Future Work.....	182
Appendix A – Chapter 2	185
A.1 Papers included in the review	185
A.2 Collected Data for RQ _{2.2}	189
A.3 Collected Data for RQ _{2.2}	191
A.4 Primary Studies Quality Assessment.....	193
Appendix B – Chapter 6	195
B.1 Repeated Measures Design (RQ _{2.2}).....	195
B.2 Repeated Measures Design (RQ ₃).....	195
Appendix C – Chapter 7	195

C.1 Data Collection Instruments	195
C.2 Default Cost-Benefit Models	217
References	220
Index	233

Acknowledgements

The current dissertation is the result of a challenging 7-years procedure and could not have been accomplished if I were not surrounded by many special people. Now that I am completing this project, I feel the need to thank the people whose assistance was a milestone in the completion of this PhD project.

First of all, I need to express my special thanks and sincere gratitude to my supervisors, Professor Paris Avgeriou and Professor Alexandros Chatzigeorgiou. Prof. Avgeriou was the person who inspired my interest in Technical Debt research, whose guidance throughout my research was priceless, and whose expertise was invaluable in the formulation of the problem statement and the completion of this dissertation. Prof. Chatzigeorgiou was the person who was close to me and helped me at the beginning of this procedure, who was always enthusiastic about the research progress, and whose insightful feedback pushed me to refine my work and bring it to a higher level. Thank you both for believing in me and giving me the chance to pursue my dream, for supporting me, and for your patience whenever my work obligations did not allow me to be as research-devoted as I should be.

The most exciting part of my PhD project was that I had the chance to collaborate with a person I highly admire, respect, and appreciate, a person I am proud of, a person that has been leading my steps during the last seven years, a person without whom I would not have been able to complete this research, and without whom I would not have made it through my PhD. I wish the words “thank you” could express the deep gratitude I feel for my brother, Apostolos Ampatzoglou. Thank you for motivating me, supporting me, guiding me, spending your time with me, shouting at me, encouraging me, tolerating me, throughout my PhD. This dissertation would not have been accomplished without you.

Of course, my research would not have been possible without the collaboration with many distinguished members of the academic community. Therefore, I would like to pay my special regards to Prof. P. Abrahamsson, Prof. E. Angelis, Prof. A. Martini, Prof. N. Mittas, Prof. K. Systä, and Prof. U. Zdun. It has also been a pleasure to work with very interesting people in both SEARCH group in RUG and SDE lab in UoM. Alexandros, Angeliki, Christina, Christos, Daniel, Dimitris, Elvira, George, Nikos, Sara, Sofia, Thodoris A., Thodoris C., Thodoris M., thank you all for your collaboration and support.

Nevertheless, working on a PhD project requires a lot of time, effort, and commitment, and I could not have reached this accomplishment without moral support and encouragement. So, first of all, I wish to acknowledge the support and infinite love of my dearest parents, Prodromos and Evangelia, who have always been next to me, supporting me by any means,

trusting me, encouraging me to follow my dream, and tolerating my frustration. Thank you, I could not have done this without you. Last but not least, special appreciation to my beloved family and friends, for their encouragement, motivation, patience, and endless supportive conversations. Lemonia, Evangelia, and Giannis, I am grateful for your great love and never-ending support. Angela, Chara, George, Maria A., Maria P., and Rallis, thank you for being there, caring for me, listening to me, understanding me, and encouraging me. You have all made this PhD thesis come true.

Chapter 1 - Introduction

The metaphor of Technical Debt (TD) was introduced in the software engineering literature in 1992 by Ward Cunningham. Cunningham used this metaphor to emphasize the trade-offs between delivering immature software code and shrinking product time to market, by parallelizing “*shipping first time code*” with “*going into debt*” (Cunningham, 1992). In the following decades, the metaphor of TD became very prominent, attracting the interest of the research community, as well as of the industry. The main selling point of the metaphor was its ability to act as a means of communication between technical and management stakeholders, denoting the need for improving the software’s internal quality (Kruchten et al., 2012).

The most recent and popular definition of TD, emerged during the Dagstuhl seminar of 2015: “*In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability*” (Avgeriou et al., 2016). In the rest of this chapter, we first present background information on the main TD concepts (see Section 1.1), then we discuss the field of Technical Debt Management (see Section 1.2), and we finally conclude with the presentation of the research design for the whole thesis (see Section 1.3).

1.1 Technical Debt Concepts

The foundations of the TD metaphor lie on two concepts borrowed from the concept of debt in economics, namely *principal* and *interest*. A software organization saves effort, considered as the ***Principal*** (of the loan) by producing immature software artifacts (e.g., design or source code) and pays the penalty for these saved funds as additional maintenance effort (***Interest***).

In Figure 1a, we visualize the relation between these two concepts, to facilitate the interpretation of the basic TD terminology, based on the study of Chatzigeorgiou et al. (2015). For any “*actual*” system, the y-axis depicts the level of design-time quality. The distance between the actual quality and the “*optimal*” quality is also denoted¹. The effort required for the development team to reach optimal quality, represents *TD Principal*: that is the vertical distance between actual and optimal quality. *TD Interest* is a side-effect of TD

¹ We note that the term “optimal” is used for simplicity, since software design optimization is a non-finite problem, for which no unanimous optimal solution can be found

Principal and represents the additional effort needed to maintain the software in the *actual* state, compared to the effort that would be needed if the system was of *optimal* quality.

Figure 1a. TD Terminology Visualization (Chatzigeorgiou et al., 2015)

SonarQube is considered to be the most frequently used tool for estimating *TD Principal* (Yli-Huumo et al., 2016), at least for Code TD, although other tools, such as Software Analysis Toolkit (SAT) of SIG have been proposed in literature (Nugroho et al., 2011). In SonarQube, TD Principal is assessed through the number of inefficiencies in the source code, and the amount of time required to fix these inefficiencies. The algorithm of the platform initially relied on a version of the SQALE method (Letouzey and Ilkiewicz, 2012), in which TD is calculated as the sum of remediation costs for all quality issues.

On the other hand, *TD Interest* is more difficult to estimate, as it refers to the difference in effort required between maintaining the actual and a hypothetical optimal system. Despite the fact that TD Interest cannot be quantified with certainty, in the literature we can identify several approaches for assessing it—most of them related to maintainability². Seaman and Guo (2011) suggest that technical debt should be repaid to “...avoid “interest” in the form of decreasing maintainability”. Conejero et al. (2018) recognize that “maintainability is one of the main characteristics contributing to Technical Debt interest”. Similarly, Zazworka et al. (2014) use defect-proneness and change-proneness as proxies for TD Interest and acknowledge their connection to future maintenance costs. McCormack and Sturtevant (2016) analyze the relationship between design decisions and maintenance costs, considered to represent TD Interest, while Kazman et al. (2015) focus on the relationship between architecture roots, i.e., defective structures, and their consequences in terms of higher maintenance costs. Under this perspective, metrics that assess

² Maintainability represents how easily a software engineer can apply changes in a specific software system (ISO/IEC/IEEE 14764-2006)

maintainability by quantifying design-time characteristics, such as inheritance, cohesion, coupling, complexity and size, can be used as proxies for TD Interest estimation. Nevertheless, its actual quantification (not through a proxy) still remains an open research direction.

1.2 Technical Debt Management

Maintenance can be considered as the costliest activity in software development, since it constitutes 50 – 75% of the total effort spent during the software lifecycle (van Vliet, 2008). In practice, some maintenance activities, such as the implementation of new functionality, the correction of errors, the enhancement of run-time qualities, or the adaptation to new environments, cannot be ignored or postponed. On the other hand, maintenance activities related to the improvement of design-time quality (proactive maintenance) are usually neglected or deferred to future iterations, so as to reduce product time to market and diminish short-term costs. Nevertheless, software systems evolve profoundly over-time and *their quality will gradually decay* (Parnas et al., 1994), if developers neglect maintenance activities, such as refactorings, resolution of bad smells, or reverse engineering. This can lead (intentionally or unintentionally) to the creation of a financial overhead, that is now popularly called Technical Debt. This overhead can become so large if TD is not properly managed, that it can eventually bankrupt software development organizations (Avgeriou et al., 2016).

According to Li et al. (Li et al., 2015) effective Technical Debt Management (TDM) consists of eight activities: **repayment** (i.e., reducing the amount of accumulated TD—e.g., apply the *Extract Methods* refactoring), **identification** (i.e., spotting the artifacts that suffer from TD—e.g., identify classes that contain *Long Methods*), **measurement** (i.e., quantifying TD in terms of both TD principal and interest—e.g., assess the *time required* to apply the Extract Method refactoring), **monitoring** (i.e., recording and assessing the evolution of TD—e.g., assessing the *contribution of new code on TD* accumulation), **prioritization** (i.e., identifying the TD items that will be repaid first—e.g., identify *classes that are very change prone*), **communication** (i.e., explaining the nature, roots or consequences of TD to stakeholders—e.g., discuss on *increasing refactoring budget during the sprints*), **prevention** (i.e., avoiding the accumulation of additional TD—e.g., apply *traceability approaches to avoid outdated documentation*), and **representation / documentation** (i.e., recording all decision, actions, metrics, related to TDM—e.g., visualize a time series of the evolution of TD over time). These eight activities can be grouped into four high-level ones (Arvanitou, 2019), according to their goal, as depicted in Figure 1.2.a.

Figure 1.2.a: Grouping of TDM activities (Arvanitou et al., 2019)

We note that despite its negative consequences (Zazworka et al., 2011), Technical Debt is often desirable, e.g., in cases when companies prefer to invest on the development of a new product / feature / service to achieve a business goal, rather than improving the quality of an existing system. As noted by Eisenberg (2013), the complete repayment of TD is considered unrealistic. To this end, Technical Debt should be continuously monitored and managed, since it is accumulated and “*hurts*” all development activities, from requirements engineering to the architectural design, all the way to the implementation (Kruchten et al., 2012). From all types of TD, code TD is reported as the most frequently studied type in research (Alves et al., 2016) and a very crucial type in the industry (Ampatzoglou et al., 2016).

1.3 Research Design

In Section 1.3.1 we discuss the problem statement that is addressed in this thesis. Next, in Section 1.3.2 we present the employed research methodology, whereas in Section 1.3.3 we discuss the research questions that the thesis deals with. Finally, in Section 1.3.4 we describe the used empirical research methods, and in Section 1.3.5 we conclude with an overview of this research

1.3.1 Problem Statement

The metaphor of Technical Debt by definition (and origin) lies in the intersection between the fields of software engineering and economics, as it uses financial concepts (e.g. debt, principal and interest) to describe software engineering phenomena (e.g., maintenance effort, design-time quality). The case of borrowing terms and ideas from other disciplines is not novel in software engineering and research in this path has produced many publications, e.g., the work of Mens et al. (2014) who discuss the application of ecology concepts. However, one should not neglect the inherent limitations of these practices (Hookes, 2009).

Therefore, in this thesis, we explore if the concept transfer from economics can prove **beneficial for our understanding of the TD phenomenon and the extent to which economic theories have been exploited so far**³.

In particular, TD research focuses mostly on the technical perspective, e.g., refactoring approaches (Fowler, 2018), structural metrics as proxies of TD Principal and TD Interest (Riaz et al., 2009), identification of bad smells (Tsantalis et al., 2008). This leads to three specific problems in TD management. First, most of the **existing methods and tools** that perform **TD quantification**, focus on providing proxies for TD Interest (e.g., maintainability metrics (Kazman et al., 2015), structural proxies (Kosti et al., 2017), defect proneness (Zazworka et al., 2014).) **without (accurately) monetizing TD interest** (Avgeriou et al., 2020) according to its definition (see Chapter 1.1), leading to various shortcomings as presented below (e.g., communication barriers, etc.). This means that we are not actually measuring TD interest per se. Second, one of the main premises of the concept of financial debt is an intrinsic link between principal and interest (through the interest rate); **in TD we have not yet explored this fundamental relation**, thus we may be missing out on exploiting this relation to estimate principal and interest. Third, TD management focuses predominantly on TD Principal and Interest, thus **neglecting others costs and financial benefits** related to TD (e.g., additional gains from reaching the market earlier, or providing better support due to quicker maintenance); such costs and benefits are of paramount importance in making informed decisions related to TD and particularly looking at TD management as a financial investment. The above can be summarized in the following problem statement:

Current TD management approaches are incomplete as they fail to take into account the financial aspects of the metaphor. This is particularly manifested through: (a) a lack of approaches and tools for monetizing TD Interest, following the actual definition of TD Interest; (b) a lack of establishing a relation between TD Interest and Principal; and (c) a lack of approaches that consider costs and financial benefits beyond TD Principal and Interest in the decision-making of TDM.

Some indicative consequences of the aforementioned problem in TD research and the application of the TD metaphor in practice, are outlined as follows, organized based on the four high-level activities defined in Chapter 1.2:

Communicating TD: The lack of a monetized assessment of TD Interest (in currency or effort), hinders the communication with business stakeholders. Although the software engineering literature on effort estimation is extensive, the current state of the art does not offer an estimation of the additional effort required to maintain a system (due to its sub-optimal quality), namely TD Interest. In particular, while the required investment for TDM

³ A literature review on how economics has been used in TD literature is provided in Chapter 2.

(Principal) is understood, its consequences (Interest) are underestimated, since they cannot be translated to money (usually presented as maintainability metrics or numbers of rule violations). This also means that a significant aspect of TD related to costs and benefits cannot be communicated with the relevant stakeholders, hindering decision making on where to spend resources. Finally, the lack of the relation between TD Principal and Interest, threatens the validity of the metaphor per se, in the sense that the financial concepts do not map to the technical concepts.

Quantifying TD: The lack of monetized TD Interest approaches, entails an inadequate quantification of one of the two cornerstones of the metaphor. Also, the inability of current practices to consider important financial aspects (e.g., obtained market share), while quantifying the cost and benefits of TD, renders them incomplete for TD-related decision making. Finally, the lack of evidence on the relation of between TD Principal and Interest does not allow to define (and quantify) a basic concept of financial debt: namely interest rate. A consequence of this lack is that a large number of financial theories are not applicable to TD, in the sense that they rely on interest rate.

Prioritizing TD: In the literature, TD prioritization relies on principal and interest assessment. However, if TD interest is only partially quantified, the prioritization index will be far from optimal. Additionally, even if this is overcome, there are no prioritization methods that consider additional costs and benefits of a repayment activity, other than principal and interest. Such approaches are neglecting important factors, that have been traditionally considered as fundamental to TD, such as the benefit that can be gained by going to the market earlier or the effort saved by selecting a quick-and-dirty solution.

Reducing TD: TD prevention is also hindered, due to the lack of a monetized assessment for interest. For instance, if we could correctly estimate interest, we would emphasize the prevention of the introduction of new TD items that are prone to produce high interest. Such approaches would also be more complete, if they took into account additional costs and benefits from preventing TD accumulation.

1.3.2 Design Science as Research Methodology

In this section we present the research approach that has been used, namely *Design Science* (Wieringa, 2009)—as outlined in Figure 1.3.2.a.

Figure 1.3.2.a: Research Methodology Outline (Wieringa, 2009)

The main components of the Design Science methodology are: “Practical Problems” and “Knowledge Questions”. A practical problem is defined as “*a difference between the way the world is experienced by stakeholders and the way they would like it (the world) to be*”. A knowledge question is “*a difference between current knowledge of stakeholders about the world and what they would like to know*”. A practical problem is usually solved by engineering cycles (Wieringa, 2009), whereas knowledge questions are usually answered by applying empirical methods. For example, in the TD domain, a practical problem could be: “*How can TD Interest be quantified in a monetary form?*”. The above problem is a practical problem, in the sense that it is solved by proposing a novel approach (probably supported by a tool) to calculate TD Interest. However, it also implicitly entails at least two knowledge questions: “*What are the factors that should be considered in the approach?*” and “*How accurate is the outcome of the approach?*”. These are knowledge questions, since: (a) they can be answered empirically (e.g., through a literature review and a case study/controlled experiment, respectively); and (b) because they enhance the body of knowledge on the specific domain. Design science imposes the use of research iterations—termed as design cycles (Hevner, 2007)—in the sense that the researcher starts from a problem statement, extracts and analyzes the corresponding practical problem, proposes a solution, evaluates the solution, and then starts over again, or digs even further by achieving explanatory goals (rather than exploratory ones). It is common that a problem is decomposed into other practical problems and knowledge questions, until the decomposition reaches the state of answerable questions, and solvable problems. The design science framework is particularly suitable for describing long-term research efforts, such as a PhD project, since it enables the tight coupling of research questions and solutions, following the evolution of the research project.

1.3.3 Practical Problems and Knowledge Questions

In this section, we present the practical problems and knowledge questions addressed in this thesis, as well as the links between them. Figure 1.3.3.a depicts the problems and questions: grey boxes represent knowledge questions and white boxes represent practical problems. Moreover, thick arrows denote sequence whereas thin arrows denote decomposition. We refer to both practical problems and knowledge questions as research questions. The main research questions are labelled with numbers from one to five. The research sub-questions are numbered with lowercase letters. Driven by the fact that the problem statement defined in Section 1.3.1 is decomposed into three distinct sub-problems, to address it we have followed four parallel research directions (one for the exploration of the state-of-the-art and -practice and one for each sub-problem). These research directions have not been pursued in a parallel manner, but sequentially. The numbering of the research ques-

tions in Figure 1.3.3.a is chronological, since input was provided from one research question to the other in this order.

As a first step towards addressing the problem statement of Section 1.3.1, we have ex-

Figure 1.3.3.a: Research questions addressed in this research PhD thesis

explored the state-of-research on the financial aspects of technical debt management. We were already aware that there was not much research work on the use of financial methods in existing TDM methods. However, in order to advance the state of the art, we needed to know exactly what has been published in this field to understand what is already achieved and which problems are brought to light. In particular, we have performed a *literature review (RQ₁) on the financial aspects in TDM*, guided by the following questions: “*What financial concepts are used?*”, “*What financial approaches are used?*”, and “*What SE technologies are mapped to financial concepts?*”.

The results of RQ₁ were insightful in many aspects, but literature does not always reflect what happens in industrial practice. Thus, as a second step, we have **explored the state-of-practice (RQ₂)**, by performing a survey on software practitioners from various countries

and companies. The survey aimed at answering the following questions: “*When is technical debt management necessary?*”, “*What types of TD are the most important ones?*”, and “*How does TD compare to other quality attributes?*”?

Upon completing the state-of-the-art and state-of-practice analysis, we had obtained a solid basis on both the current research approaches and open research questions, as well as industrial TD practices. Subsequently, we worked on the first sub-problem of the problem statement (i.e., interest monetization): How can ***TD Interest be Quantified (RQ₃)***? To solve this problem, we applied a 3-step approach. First, we produced the theoretical framework, upon which the actual interest calculation is based and introduced the term of TD sustainability (RQ_{3.a}). TD sustainability refers to how long cumulative interest stays lower than principal and is based on findings from RQ₂ (namely that TDM is more useful for projects with long maintenance period ahead). Next, we proposed an interest theory—termed FITTED (*Framework for Managing Interest in Technical Debt*); and instantiated it at the source code level (RQ_{3.b}). We opted to focus the calculation of interest at the code level based on the answers to RQ₁ and RQ₂, which rank code TD as one of the most important types in both industry and academia. Finally, we validated FITTED in an industrial setting (RQ_{3.c}).

As a next step, we explored the second sub-problem: whether ***principal is related to interest (RQ₄)***, which is one of the fundamental notions in the theory of financial debt. Therefore, we asked three knowledge questions: “*Is TD Principal related to TD Interest?*”, “*What aspects of TD Principal are related to TD Interest?*”, and “*What aspects of TD Interest are related to TD Principal?*”. Having established the FITTED framework as a result of RQ₃, we are able to quantify interest, and subsequently explore its relation to principal. The results confirmed the relation between TD Principal and TD Interest, thus strengthening the validity of the TD metaphor, which was challenged by the lack of evidence on the existence (even indirect) of interest rate (Schmid, 2003). To answer RQ₄, we have used the Mantel test, which was originally introduced in medical science and ecology.

As a final step of this PhD project, we have dealt with the third part of the problem statement: i.e., the lack of approaches that consider costs and financial benefits beyond TD Principal and Interest in the decision-making of TDM. To this end, we proposed ADMIT (*Architectural Decision-Making as Financial Investment*), which considers the ***costs and financial benefits in decision-making (RQ₅)***, enabling the unified assessment of technical factors, such as TD Interest and TD Principal, with business factors (e.g., market share) that are considered related to TD accumulation. We instantiated ADMIT at the architecture level, which according to RQ₂ is an important type of TD in industry. The decision to switch from code TD to architecture TD at this level was made, since architecture is by definition closer to software costs and business qualities. To validate ADMIT, we have

performed an industrial case study at the embedded systems domain, which revealed the benefits that can be obtained by ADMIT, as well as its industrial relevance.

1.3.4 Using Empiricism to Answer Knowledge Questions

Empirical Software Engineering uses experiences and/or observations to retrieve evidence from a real-world context or an artificial setting suitable for investigating a phenomenon of interest (Tichy and Padberg, 2007). In this thesis, we have used predominantly the case study method.

Case studies investigate “a contemporary phenomenon within its real-life context, especially when the boundaries between phenomenon and context are not clearly evident” (Yin 2003). They are used for monitoring real-life projects, activities or assignments. In case study research, usually different data collection methods are used. The goal is to seek convergence of evidence (from multiple, complementary data sources), a process that is often called triangulation. The case study is normally aimed at tracking a specific attribute or establishing relationships between different attributes (Wohlin et al., 2012). Regarding data collection, we used three methods (Lethbridge et al., 2005): (a) analysis of work artifacts; (b) interview and questionnaires; (c) and focus groups. The subjects of our case studies were either open-source projects, industrial projects, or practitioners (in human-centric designs).

To explore the literature, we have used the **Systematic Literature Review (SLR)** method. SLRs use data from previously published studies for the purpose of research synthesis, which is the collective term for a family of methods for summarizing, integrating and, where possible, combining the findings of different studies on a topic or research question. Such synthesis can also identify crucial areas and questions that have not been addressed adequately with past empirical research. It is built upon the observation that no matter how well-designed and executed, empirical findings from single studies are limited in the extent to which they may be generalized (Kitchenham et al., 2009).

Finally, to answer RQ₂, we employed the **survey** method. Surveys rely on the existing knowledge of experts on a topic, method, or tool that is used for a long period of time. Given their expertise on the topic, survey participants can answer questions in an unsupervised way. Given the completely uncontrolled nature of the method, responses from various participants are required (Kitchenham and Pfleeger, 2002). Also, given the low response rate of participants, a large audience must be reached before conducting the study.

An overview of the empirical research methods that were used for answering each knowledge question is provided in Table 1.3.4.a. We note that for the special case of secondary studies, the collection of data relied on the inspection of the corresponding manu-

scripts from the author and the recording of values for the predetermined variables (based on the secondary study protocol).

Table 1.3.4.a: Empirical methods used to answer the knowledge questions

Code	Knowledge Question	Empirical Method	Data Collection	Subject	Section of Study Design
RQ ₁	What are the financial aspects of TD in literature?	SLR	-	Literature	2.4
RQ ₂	How do software engineers perceive TD?	Survey	Questionnaires	Practitioners	3.3
RQ _{3.a}	Can we produce a theoretical framework for TD Interest calculation?	SLR	-	Literature	4.3
RQ _{3.c}	Is the proposed approach accurate?	Case Study	Mining Software Repositories	Industrial	5.4
RQ ₄	What is the relation between TD Interest and TD Principal?	Case Study		Open-source	6.4
RQ _{5.b}	Is the proposed approach valid?	Case Study	Interviews Questionnaires Focus Group	Practitioners	7.5

1.3.5 Overview of the Dissertation

The main body of this dissertation contains six chapters. Table 1.3.5.a presents the knowledge questions, the practical problems, and the chapters, in which they are addressed.

Table 1.3.5.a: Overview

Knowledge Question / Practical Problem	Chapter
KQ ₁ : What are the financial aspects of TD in literature?	Chapter 2
KQ ₂ : How do software engineers perceive TD?	Chapter 3
KQ _{3.a} : Can we produce a theoretical framework for TD Interest calculation?	Chapter 4
PP _{3.b} : Propose an approach for quantifying TD Interest KQ _{3.c} : Is the proposed approach accurate?	Chapter 5
KQ ₄ : What is the relation between TD Interest and Principal?	Chapter 6
PP ₅ : How can we consider costs and benefits in the decision making of TD?	Chapter 7

Chapters 2 to 7 are based on scientific journal or conference articles that are either published or accepted for publication. In all the publications, the PhD student was the first author and main contributor; other authors include the 2 supervisors as well as academic and industrial collaborators. In the following, each chapter is briefly outlined:

- Chapter 2 is based on a paper published in Information and Software Technology (IST) (Ampatzoglou et al., 2015a). The study is one of the first secondary studies in TD and the first (and until now the only one) focusing on the financial aspects of TD. The study has by now received a significant amount of citations. IST is one of the top venues in the software engineering community, whereas the study itself *was among the most downloaded ones* for two months.
- Chapter 3 is based on a paper published in the 8th International Workshop on Managing Technical Debt (MTD 2016) (Ampatzoglou et al., 2016a). This study provides an overview of the perception of software engineers on technical debt, providing an overview of the state-of-practice on the topic. The survey reached practitioners from 5 countries and 7 companies.
- Chapter 4 is based on a book chapter published in the Lecture Notes in Business Information Processing (LNBIP) (Ampatzoglou et al., 2016b). This study has a strong financial background, since it tailors existing interest theory approaches in the context of TD. The chapter was an extended and revised version of a conference paper, submitted upon editorial invitation.
- Chapter 5 is based on a paper published in the 1st International Conference on Technical Debt (TechDebt 2018) (Ampatzoglou et al., 2018). In this study we evaluated FITTED in an industrial context, by performing a case study in collaboration with industry from the enterprise application domain. TechDebt is the premiere scientific conference on TD.
- Chapter 6 is based on a paper published in Information and Software Technology (IST) (Ampatzoglou et al., 2020a). This study explored the relation between TD Principal and TD Interest, and provided empirical evidence on the existence of this relation. While this relation is expected to be prominent in any kind of debt, it was not explored (in a systematic manner) until this study. To prove the existence of this relation we performed a case study on 10 open-source projects.
- Chapter 7 is based on a paper accepted for publication at Information and Software Technology (IST) (Ampatzoglou et al., 2020b). The paper proposed the ADMIT approach and tool for treating architectural decisions as financial investments. The approach can be fed with various technical (e.g., TD Principal, TD Interest) and business (e.g., market share, customer support) parameters as inputs and treat them in a unified manner. This approach monetizes various technical characteristics of software, so as to be easily understood by managerial stakeholders and compares, in a uniform way, parameters that do not match in nature and unit of measurement.

Based on: Ar. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou — “*The Financial Aspect of Managing Technical Debt: A Systematic Literature Review*”, Information and Software Technology, Elsevier, 64, pp.52-73, 2015.

Chapter 2 – The Financial Aspect of Managing Technical Debt: A Systematic Literature Review

Technical debt is a software engineering metaphor, referring to the eventual financial consequences of trade-offs between shrinking product time to market and poorly specifying or implementing a software product, throughout all development phases. Based on its interdisciplinary nature, i.e., software engineering and economics, research on managing technical debt should be balanced between software engineering and economic theories. The aim of this study is to analyze research efforts on technical debt, by focusing on their financial aspect. Specifically, the analysis is carried out with respect to: (a) how financial aspects are defined in the context of technical debt, and (b) how they relate to the underlying software engineering concepts. In order to achieve the abovementioned goals, we employed a standard method for SLRs, applied it on studies retrieved from 7 general-scope digital libraries, and selected 69 studies relevant to the financial aspect of TD. The most common financial terms that are used in technical debt research are Principal and Interest, whereas the financial approaches that have been more frequently applied for managing technical debt are real options, portfolio management, cost/benefit analysis and value-based analysis. However, the application of such approaches lacks consistency, i.e., the same approach is differently applied in different studies, and in some cases lacks a clear mapping between financial and software engineering concepts. The results are expected to prove beneficial for the communication between technical managers and project managers, in the sense that they will provide a common vocabulary, and will help in setting up quality-related goals, during software development. To achieve this, we introduce: (a) a glossary of terms, and (b) a classification scheme for financial approaches used for managing technical debt. Based on these, we have been able to underline interesting implications for researchers and practitioners.

2.1 Motivation

One of the most prevalent characteristics of technical debt is its interdisciplinary nature, since it combines elements from both *financial and software engineering theory*. Although this nature may lead to additional challenges (resulting from the gap between software and financial perspectives) it can potentially help the *communication between both practition-*

ers and researchers. Concerning practitioners, we refer to the communication gap between technical stakeholders (software engineers, architects, testers etc.) and project managers. On the one hand, project managers are interested in concepts like value, cost, benefit, debt, principal (i.e., capital), interest, etc. Moreover, they are not particularly focused on the design-time quality of intermediate artifacts during the software development lifecycle, since it is only indirectly associated with their basic goals as stakeholders in the software development lifecycle, i.e., increase benefit, decrease production cost, shrink time to market etc. On the other hand, technical stakeholders are usually more focused on design-time quality. Enhanced design-time quality in terms of e.g., maintainability, reusability and understandability, eases the post-production activities of development teams and decreases the effort needed for the development of similar projects. However, in practice, high-level budget and effort allocation decisions are taken by project managers. Thus, in order for maintenance activities to be approved, technical stakeholders should communicate their benefits to project managers. In this type of communication, the terminology of technical debt can prove beneficial. *Practitioners' experience suggests that using economics-based terminology and approaches* like real-options, cost/benefit analysis, portfolio management, etc., bridges the gap between software engineers and managers and *facilitates the negotiation of trade-offs* of quality for quicker product delivery (Eisenberg, 2013).

Despite the communication benefits that Technical Debt may bring, due to its aforementioned interdisciplinary nature, the *body of knowledge* on the subject can be rather difficult to comprehend in-depth, since it requires expertise or at least experience from two rather diverse scientific domains (i.e., software engineering and economics). Therefore, researchers with financial background are often not sufficiently familiar with terms like refactorings, source code analysis, etc., whereas software engineering researchers are usually not experts in applying methods like real-options, portfolio management, etc. Consequently, in many cases, research on the subject: (a) *uses ambiguous terminology and sometimes misuses terms*, (b) *is not balanced* between the economical and the software engineering aspects, or in other words it is not truly interdisciplinary. For example, when a purely software engineering team with limited expertise on economics, apply the real-options theory, there is an increased possibility that the method is not perfectly applied.

Furthermore, the current literature in TD lacks a study summarizing the state of the art and practice with respect to financial approaches used in TD. This makes it difficult for practitioners to search for such approaches, and evaluate them for fitness of purpose and applicability for the particular problem at hand. Even when they manage to select an appropriate approach, applying such an approach can be challenging, since most of them lack an established way of mapping their original (from the financial domain) inputs and outputs to

software engineering technologies⁴. For example, in real options analysis the improvement probability factor is required as an input to the method (Reilly and Brown, 2012), but how is this probability calculated in the context of technical debt and what software engineering technologies should be used for assessing it, and how? Moreover, the lack of a state of the art also hinders researchers in understanding the overall picture of what has been studied by this community, and what is still missing.

In this study we will analyze existing literature, by employing the Systematic Literature Review (SLR) method (Kitchenham et al., 2009) to tackle the two aforementioned problems, i.e., the communication gap among researchers and among practitioners, and the lack of a study summarizing the state of the art and practice on financial approaches for technical debt management. Therefore, this study has two goals:

- (goal-a)** Introducing a glossary of financial terms and definitions related to technical debt management.
- (goal-b)** Classifying financial approaches (originating from traditional or software economics) used in technical debt management.

By referring to Technical Debt Management in this study, we are interested in all TDM activities as proposed by Li et al. (2015), i.e., (a) identification, (b) measurement, (c) prioritization, (d) repayment, (e) monitoring, (f) prevention, (g) communication, and (h) representation/documentation. However, since the focus of the study is on the financial perspective of technical debt, special attention will be given to the measurement (or quantification) activity. We note that due to the nature of technical debt research, we have broadened the primary study search space to generic computer science literature, rather than focusing on software engineering literature only.

The rest of the chapter is organized as follows: in Chapter 2.2, we present related work, i.e., other studies that aim at providing an overview of the research state of the art on the domain of technical debt. In Chapter 2.3, we provide some background information on financial concepts that will be used during the presentation and discussion of the results. Next, in Chapter 2.4, we present the protocol of this systematic literature review. In Chapter 2.5 we present the raw results of this study and answer the research questions, whereas in Chapter 2.6 we discuss the basic contributions of this study w.r.t. *goal-a* and *goal-b*.

2.2 Related Work

Although the financial aspect of technical debt is a research topic that has attracted the attention of both academia and industry, to the best of our knowledge there are no literature

⁴ By the term ‘software engineering technologies’ we refer to: (a) any element of the software engineering process, e.g., tools, artifacts, activities (Kruchten, 2000), (b) quality attributes, and (c) well-established approaches that aim at the improvement of design-time qualities (e.g., patterns, refactorings).

reviews or mapping studies (either systematic or not), summarizing the research state of the art on the subject. The only two studies that have been identified as related work to this study, are a Multi-vocal Literature Review (MLR) by Tom et al. published in 2013 (Tom et al., 2013) and a Mapping Study (MS) by Li et al. published in 2015 (Li et al., 2015). The main points of deviation of our work, compared to these studies, are:

- the use of a different research method (SLR vs. MS vs. MLR), and
- the different focus of the research goals (financial aspects of technical debt vs. software engineering aspects of technical debt).

The main difference between a Multi-vocal Literature Review (MLR) and a Systematic Literature Review (SLR) or a Mapping Study (MS) is the fact that, while SLRs and MSs use as input only academic peer reviewed articles, in MLRs, grey literature, such as blogs, white papers and webpages, is also considered as input. We note that despite the differences in the used research methods, the results of both types of studies are valid; however, through a different perspective. Consequently, concerning technical debt, the multi-vocal results (Tom et al., 2013) are expected to provide indications on how the concept is exploited by both researchers and practitioners; although it goes without saying that using such not peer-reviewed experience reports or position articles raises significant reliability and validity issues (Ogawa and Malen, 1991). On the other hand, the results of an SLR or an MS could provide an established body of knowledge, focusing only on research contributions. However, comparing the research goals and designs of MS and SLR, the following differences can be identified:

- SLRs aim at providing a complete, detailed and fair synthesis of evidence related to a topic of interest, whereas MSs aim at providing an overview of a research area to assess the quantity of evidence (Kitchenham et al., 2011);
- the research objectives for MSs are usually high level and include issues, such as identification of research subtopics and used research, and classification of research themes. On the other hand, SLRs usually synthesize data in a more detailed way (Kitchenham et al., 2011).

Additionally, the goal of the three studies is completely different as well. More specifically, the studies of Tom et al. (2013) and Li et al. (2015) aimed at understanding the nature of technical debt and its implications for software development, whereas our study is focused on the financial perspective of technical debt. However, although the research questions of the three studies do not overlap, it is expected that discussing the results will provide possibilities for synthesizing the outcomes of the three secondary studies.

2.3 Background Information

In this chapter we provide background information on financial debt i.e., definitions, related terms and approaches. More specifically, we provide: (a) basic definitions on financial debt that are expected to be investigated in the related literature on technical debt, (b) broader financial terms, related to interest theory, and (c) some fundamental strategies for handling financial debt. This chapter aims at supporting readers with a software engineering background to become familiar with some basic financial concepts that will be needed for understanding the results of this SLR. All terms discussed in this chapter will be used either during the presentation of results or in the discussion of directions for future research.

Basic Financial Debt Terms: Debt is a term used to describe the amount of money owed by one party (*debtor or borrower*) to another party (*creditor or lender*). The certain amount of money derives from a loan, which denotes that the money has been lent by the creditor to the debtor for a specific period of time. The obligation of the debtor is to repay a larger sum of money to the creditor at the end of that period (Mishkin and Eakins, 2012). The original amount of money borrowed is called the *principal*, while the additional amount paid back constitutes the *interest*. To accomplish this agreement, the borrower issues a *debt instrument*, which specifies the type of the loan and obliges the debtor to repay it at the *maturity date* (date of repayment). Economists consider the interest to be a fee that is charged for the use of money, i.e., the creditor is being remunerated for the undertaken risk of losing the principal and for the lost benefit that another *investment* of the money would have yielded (*opportunity cost*). On the other hand, the debtor (individual, company or government) is in need of money, in order to proceed with an investment or to cover temporary lack of *liquidity*, therefore decides to pay the interest in the future.

Interest is calculated as a percentage of the principal, usually on an annual basis, namely *interest rate*. Interest rate is dependent upon systemic financial factors (e.g., *quantity of money* and *price level* (Friedman, 1968)) and the debtor's *risk rating* (Mishkin and Eakins, 2012). If the debtor is assessed as a low-risk partner, a lower interest rate shall be applied. On the other hand, if the borrower is considered to be of high-risk, a higher interest rate will be charged (see chapter *Managing Debt Strategies* below, on risk management). Interest can be discriminated into two common types (simple and compound), according to the way it is calculated. When interest is calculated in every period on the amount of the principal, it is called *simple interest*. In this case, the total amount of interest increases linearly. On the other hand, *compound interest* occurs when it is calculated on the principal plus the accumulated interest. Thus, the total amount of interest increases exponentially (Kellison, 2008).

Broader Financial Terms (related to investments and interest theory): In order to elaborate on concepts of debt and loan we present some additional terms that are considered important for financial transactions. More specifically, *liabilities* and *assets* are terms used in both financial and accounting literature and practice and refer to elements that constitute obligations or create economic value for their owners respectively. Moreover, in order to compare the value of a sum of money or a debt instrument through different periods of time, the concepts of *present value* (PV) and *future value* (FV) are used. So as to assess the effectiveness of an investment, the *Return on Investment* (ROI) ratio is used. A high level of ROI indicates an efficient investment. *Financial leverage* is perceived as the use of *loanable funds*, i.e., money derived from loans by a company, in order to invest in actions that will result in profit. Specifically:

- An asset could be a company's plant, mechanical equipment or money, while, a company's liabilities can consist of its loans, accounts payable or any other company's obligations. Referring to finance, every debt instrument is a liability for the debtor and at the same time it is an asset for the creditor (Mishkin and Eakins, 2012).
- Present Value or Present Discounted Value (PV) refers to the current worth of an amount of money paid or received in the future. Present value is calculated as in (1), where r represents the given rate of interest and n stands for the number of periods (Mishkin and Eakins, 2012)

$$PV = \frac{FV}{(1+r)^n} \quad (1)$$

- Future Value (*FV*) or *time value of money* refers to the value of a present sum of money in a specific time the future (Kellison, 2008). According to Kellison (2008), future value is calculated as:

$$FV = PV * (1 + r)^n \quad (2)$$

- Financial leverage increases the undertaken risk; nevertheless, it is generally acceptable and sometimes preferable over other forms of financing by shareholders, as it increases the market value of the *shares* (Duffie and Singleton, 2003). The degree of a firm's leverage expresses its dependence on long term debt. This dependency can be shown by the debt to equity ratio that is the percentage of debt on the company's equity.
- Return on Investment (*ROI*) is defined as a ratio that measures the return of an investment divided by the invested capital. More specifically, ROI is defined in (3), where Net Profit is the total gains of the investment minus the cost of the investment (Farris et al., 2010). Besides the evaluation of a single investment, ROI ratio can be used to compare different investments.

$$ROI(\%) = \frac{\text{Net Profit}}{\text{Cost of Investment}} \times 100 \quad (3)$$

Managing Debt Strategies: Decision-making on undertaking an investment or on taking a loan so as to perform an investment should consider the risks and the cost of the specific financial transaction. Thus, in order to make the most profitable decision, economists usually apply the *Cost/Benefit Analysis* (CBA), i.e., a technique that compares the cost of the realization of a decision to the expected benefit of this action. The implementation of CBA method should take into account all the possible expected costs until the completion of the project, such as cost of acquisition, man hours needed for implementation, operational costs etc.

As mentioned above, risk is another factor that should be considered during the decision making on the realization of a financial transaction. The risk concerning a financial transaction may occur due to *interest rate fluctuations* or to other factors involving the two parties of the transaction (Mishkin and Eakins, 2012), e.g. the probability of the debtor to *bankrupt*. In order to accomplish their goals, companies should apply *risk management* strategies. More precisely, they should: (a) examine the possible sources of their risk, (b) evaluate the probability of its occurrence, and (c) implement methodologies for measuring the credibility of their counterparties (Duffie and Singleton, 2003).

A specific technique that is used for reducing or eliminating financial risk and reducing market uncertainty is *hedging* (Mishkin and Eakins, 2012). Hedging involves the engagement in a financial transaction in order to counterbalance the risk undertaken by another transaction. For example, hedging may offset a *long position* (that is buying an asset) by taking an additional *short position* (that is selling an asset). In literature, a field that offers great opportunities for hedging is *financial derivatives market* (Mishkin and Eakins, 2012). Financial derivatives can be defined as financial products whose values are linked to the values of previously issued instruments or other variables (Hull, 2009) and (Mishkin and Eakins, 2012).

A common form of financial derivatives contract is an *Option* contract, which gives the owner the right to buy (*call option*) or sell (*put option*) a specific financial product at a specified price (*strike price* or *exercise price*) in the future. The purchaser of an option is not obligated to exercise the right of buying or selling, on or until the expiration date. On the other hand, the seller of the option is obligated to sell or buy the financial instrument, if the owner decides to do so. In order to acquire the option to buy or sell a product at a specified price, the purchaser pays an amount called a *premium* (Mishkin and Eakins, 2012).

Option Pricing Theory has been widely used during the last decades in order to evaluate opportunities in non-financial or “real” investments, such as land, buildings, plant expansion or multi-stage R&D (Borison, 2005) and (Hull, 2009). *Real Option Analysis* (ROA) is

extensively applied to assess IT investment risk. By equating managerial flexibility with real options, it permits the inclusion of uncertainty in the valuation model (Benaroch, 2002).

Finally, the total amount of assets and liabilities of a company or individual is called a *portfolio*. Portfolio management is the technique of managing the assets and liabilities in order to achieve a satisfactory benefit. The aim of a successful portfolio management is to achieve the highest possible profit and at the same time to minimize the risk of the undertaken financial transaction (Reilly and Brown, 2012).

2.4 Review Methodology

This chapter presents the protocol of this systematic literature review. A protocol constitutes a pre-determined plan that describes research questions and how the systematic literature review will be conducted. The review protocol includes six activities, namely (Kitchenham et al., 2009): (a) define research objectives and questions, (b) define search procedure (search terms and resources), (c) define study selection procedure and inclusion/exclusion criteria, (d) define study quality assessment, (e) define data extraction strategy, and (f) define synthesis of the extracted data.

2.4.1 Research Objectives and Research Questions

We reformulate the goals of this study, using the Goal Question Metric format (Basili et al., 1994), as follows: *analyze* existing literature on the financial aspects of technical debt *for the purpose* of characterization and evaluation, with respect to terminology (see goal-a in Chapter 1) and the employed financial approaches (see goal-b in Chapter 1), from the *point of view of* researchers and practitioners. In order to more systematically investigate the aforementioned goal, our study sets three research questions, which are listed here and explained right below:

- **RQ1:** *Which are the most common financial terms used in the context of technical debt management?*
- **RQ2:** *Which financial approaches have been applied in technical debt management?*
- **RQ2.1:** *Which financial approaches have been applied for measuring technical debt, in terms of financial units (e.g., man-months or US\$)?*
- **RQ2.2:** *Which financial approaches have been applied for identifying, prioritizing, repaying and monitoring technical debt?*
- **RQ3:** *Which software engineering technologies have been used by financial approaches in the context of technical debt management?*

The two goals stated in Chapter 1, i.e., glossary of terms (goal-a) and classification of financial approaches in TDM (goal-b), are achieved in two steps. First, we collect data (ac-

cording to the process in Chapter 2.4.5) in order to answer each research question and present them in Chapter 2.5. Second, we further analyze the collected data in order to achieve the stated goals, as presented in Chapter 2.6. Specifically, for the first goal we will utilize the results of RQ_1 (they will provide a list of terms that are used in technical debt research) and the background information presented in Chapter 2.3. Specifically, we use the descriptions of terms in financial theory, as presented in Chapter 2.3, in order to fine-tune the definition of the glossary terms. Additionally, for the second goal, we will synthesize the results of RQ_2 (that will provide a list of financial approaches) and RQ_3 (that will provide us information about the employed software engineering technologies), which in turn are used as inputs when applying the financial approaches in technical debt management. Consequently, the classification synthesizes the set of financial approaches with the software technologies they use as input. We note that due to our special interest on TD measuring approaches we have set a separate research question concerning measurement ($RQ_{2.1}$), whereas other managing activities are investigated through $RQ_{2.2}$.

2.4.2 Search Process

The search procedure aimed at the identification of candidate primary studies, which would be either included or excluded from the final set of the primary studies. The search plan involved automated search into seven well-known digital libraries (ACM Digital Library, IEEEExplore, ScienceDirect, SpringerLink, Scopus, Web of Science, and Google Scholar), and a manual search for string calibration and validation of the process accuracy. To cross check the results we obtained from the automatic search and to create a valid search string, we searched a small number of venues manually, similarly to determining a “quasi-gold” standard, as proposed by Zhang and Babar (2010). Venues for the manual search were determined based on their probability to publish research in the context of software engineering and technical debt. Therefore, the manual search was limited to the following⁵:

- International Workshop on Managing Technical Debt (MTD Workshop)
- IEEE Software (SW)
- Journal of Software Quality (SQJ)

When manually searching the venues, we considered title, keywords, and (if necessary) abstract, and compared the results with those of the automated search, so as to estimate if we were missing any studies. The results from the automated search included all studies found from applying the “quasi-gold” standard. In the automated search and by taking into

⁵ MTD has been selected since it is the most specialized venue for technical debt research, whereas SW and SQJ have been selected, since they are highly reputed software engineering venues (the former is a magazine while the latter a journal), and the authors are aware of both having recently published a number of articles on TD.

account the limited number of articles on the subject, as well as the natural connection of the term ‘debt’ to economics, we believe that the search string should consist of one string, i.e., “*technical debt*”. The searching completed on September 2013.

2.4.3 Article Filtering Phases

The papers that were selected as candidate primary studies in the review should be relevant to technical debt, and more specifically to its economic perspective. In line with Dybå and Dingsoyr (2008), there are three stages of filtering the article set to produce the primary study data set:

1. The search process, described in 2.4.2, returned a set of candidate primary studies.
2. The article set went through a manual inspection of each article’s title, and abstract/conclusions. The inclusion/exclusion criteria that have been used in every stage are listed below:

Inclusion Criteria:

- Study should be related to financial aspects of technical debt. In order for a study to be related to financial aspects of technical debt it should: (a) apply a financial method in technical debt management, or (b) measure financial debt in terms of some currency, e.g., US\$ (i.e., studies that provide indicators of technical debt – measures that are correlated to technical debt, but do not quantify in money or effort – are excluded), or (c) define/discuss a financial term in the context of technical debt.

Exclusion Criteria:

- Study only mentions technical debt in an introductory statement and does not focus on its calculation, definition, economical aspects etc.
 - Study is an editorial, keynote, opinion, tutorial, workshop summary report, poster, or panel. Such papers have been excluded either due to their small size or due to the fact that such articles are usually not peer-reviewed.
 - Study’s full text is not available
3. The qualified articles went through the same inclusion/exclusion criteria, by taking into account the full text of the articles.

Every article selection phase has been handled by the author of the thesis and possible doubts have been resolved by a discussion between the author of the thesis and co-authors of the article.

2.4.4 Quality Assessment

All selected papers, which are included in the review, have gone through a quality assessment process, in order to investigate if they were of adequate standard, so as to be included

in this review. Thus, each study has been evaluated against a set of questions with respect to the used method, and the quality of the reporting. As the basis for our primary study quality assessment, we used the following instrument:

1. Are the aims and objectives clearly reported (Dybå and Dingsoyr, 2008)? (0: no, 0.5 partially, 1.0 yes)
2. Is the study's focus or main focus on TD (Dybå and Dingsoyr, 2008)? (0: no, 0.5 partially, 1.0 yes)
3. How much evidence supports the results of the study (Alves et al., 2010) (0.0 – 1.0 points)?
 - Level 0: No evidence (0.0).
 - Level 1: Evidence obtained from demonstration or working toy examples (0.2).
 - Level 2: Evidence obtained from expert opinions or observations (0.4).
 - Level 3: Evidence obtained from academic studies, e.g., lab experiments (0.6).
 - Level 4: Evidence obtained from industrial studies, e.g., case studies (0.8).
 - Level 5: Evidence obtained from industrial practice (1.0).
4. Is there a clear statement on the TDM activity that study focuses? (0: no, 1.0 yes)
5. Is the study sufficiently using both the financial and the software engineering aspect of TD? (0: none, 0.5 only one, 1.0 both)

The first two questions concerning the aims and focus of the examined primary studies are based on the work by Dybå and Dingsoyr (2008). The third question evaluates the empirical rigor and industrial relevance of the studies based on the classification by Alves et al. (2010). The final two questions aim at assessing the fitness of the candidate primary study for answering our research questions. The complete evaluation of the studies is presented in Appendix A.4. The inclusion threshold has been set to a total score of 2.0, because questions 3, 4, and 5, might be assigned a value of zero in a paper that approaches technical debt management from a theoretical perspective. For such papers, we require at least that the goals and focus on TDM are explicit.

2.4.5 Data Collection

During the data collection phase, we collected data on a set of variables that describe each primary study. Data collection has also been handled by the author of the thesis and has been validated by the second co-author of the article. If both reviewers assigned the same value to one variable, this value would be assigned to the variable without further discussion. In any other case, a discussion among the authors would result in consensus about the value to be assigned. We have not applied an agreement measure as the number of re-

searchers involved in the review is not significantly large. However, all conflicts have been recorded. For every study, we have extracted the following data:

- [A1] Author
- [A2] Year
- [A3] Title
- [A4] Source
- [A5] Venue
- [A6] Type of Paper (conference / journal)
- [A7] Keywords
- [A8] Quality Score
- [A9] Focus of the research (financial, software engineering, both, neither⁶)
- [A10] Financial Terms
- [A11] Ways of Measuring Technical Debt
- [A12] Ways of Managing Technical Debt
- [A13] Software Engineering Technologies
- [A14] Limitations
- [A15] TDM Activity (Li et al., 2015)

2.4.6 Data Analysis

The mapping between variables and research questions is provided in Table 2.4.6, accompanied by the analysis methods used on the data. Variables [A1] to [A9] are not listed in this table, since they are not used for answering any specific questions, but for demographic characterization of the study. Specifically, for RQ₁, will be investigated through a frequency table of the identified financial terms. To identify if there are any dominant approaches for quantifying specific financial terms (RQ_{2.1}), we will cross-tabulate financial terms found in the selected primary studies against TD measuring approaches. Additionally, we will produce a pie chart on the ways of managing technical debt, to visualize the frequency of their application (RQ_{2.2}). Finally, to answer RQ₃, we will create a frequency table for identifying the software engineering terms that are more relevant to TDM research.

⁶ Without in-depth analysis of either the financial or the software engineering aspect

Table 2.4.6: Data Synthesis Methods

Research Question	Variables Used	Synthesis Method
RQ₁	[A10]	Frequency tables for [A10]
RQ₂	[A10]	Cross-tabulation for [A10], [A11] Pie chart for [A12] Discussion of [A14] and [A15]
	[A11]	
	[A12]	
	[A14]	
	[A15]	
RQ₃	[A13]	Frequency tables for [A13]

2.5 Results

After conducting the article searching and filtering phases, as described in Chapters 2.4.2 and 2.4.3, we have concluded in the inclusion of 69 primary studies. In Table 2.5.a, for each investigated data source, we present the number of papers that have been returned as candidate primary studies (step 1), the number of papers qualified after primary study selection on the basis of title and abstract (step 2), and the final number of primary studies (step 3). On the completion of these phases we examined the quality of each article, based on the guidelines of chapter 2.4.4, and all articles have qualified.

Table 2.5.a: Primary Studies per Digital Libraries

Digital Library	step 1	step 2	step 3
ACM Digital Library	72	19	14
IEEE Xplore	136	29	23
SpringerLink	46	4	1
ScienceDirect	33	5	2
Scopus	107	12	10
Web of Science	16	6	0
Google Scholar	763	34	19
Total	1173	103	69

Table 2.5.b illustrates the frequency of specific venues from which relevant articles have been retrieved. The most popular venue in our final set of primary studies in the Managing Technical Debt Workshop (MTD), followed by Cutter IT Journal and IEEE Software. The interdisciplinary nature of technical debt is reflected in the diversity of venues in which primary studies with reference to the financial aspect of technical debt have been identified.

Specifically, we retrieved 27 studies from TD-specific venues, 25 studies from software engineering (SE) venues, 8 studies from Information Technology (IT) venues, and 9 studies from generic (GEN) venues such as Communications of the ACM and IBM Journal of Research & Development.

Table 2.5.b: Primary Studies per Venue

Venue	Type	Frequency	Nature
Managing Technical Debt	Workshop	27 (40.2%)	TD
Cutter IT Journal	Journal	7 (10.1%)	IT
IEEE Software	Journal	7 (10.1%)	SE
Agile Conference	Conference	2 (2.9%)	SE
Software Quality Journal	Journal	2 (2.9%)	SE
Advances in Computers	Journal	1 (1.4%)	GEN
Agile Processes in Software Engineering and Extreme Programming	Conference	1 (1.4%)	SE
API Design for C++	Book	1 (1.4%)	GEN
Communications of the ACM	Journal	1 (1.4%)	GEN
Workshops of the Computer Software and Applications Conference	Workshop	1 (1.4%)	GEN
Hawaii Int. Conf. on System Sciences	Conference	1 (1.4%)	GEN
IBM Journal of Research and Development	Journal	1 (1.4%)	GEN
Int. Journal of Advancements in Computing Technology	Journal	1 (1.4%)	GEN
Int. Conf. on Building and Exploring Web Based Environments	Conference	1 (1.4%)	GEN
Int. Conf. on Evaluation and Assessment in Software Engineering	Conference	1 (1.4%)	SE
Int. Conf. on Software Engineering	Conference	1 (1.4%)	SE
Int. Conf. on Software Engineering Advances	Conference	1 (1.4%)	SE
Int. Conf. on Software Engineering and Mobile Application Modelling and Development	Conference	1 (1.4%)	SE
Int. Conf. on Software Maintenance	Conference	1 (1.4%)	SE
Int. Conf. on Software Testing, Verification and Validation	Conference	1 (1.4%)	SE
IT Professional	Journal	1 (1.4%)	IT
Object Oriented Programming Systems Languages and Applications	Conference	1 (1.4%)	SE
Procedia Computer Science	Journal	1 (1.4%)	GEN
Proceedings of the Third C* Conf. on Computer Science and Software Engineering	Conference	1 (1.4%)	SE
Quality of Software Architectures	Conference	1 (1.4%)	SE
SIGSOFT Software Engineering Notes	Journal	1 (1.4%)	SE
Symposium on Empirical Software Engineering and Measurement	Symposium	1 (1.4%)	SE
Working Conference on Software Architecture	Conference	1 (1.4%)	SE
Workshop on Future of Software Engineering Research	Workshop	1 (1.4%)	SE

As described in Chapter 2.1, technical debt is an interdisciplinary domain, which should ideally deal with aspects of both economics and software engineering. To this end, it is

interesting to examine how far the selected studies cover either the economics or the software perspective aspect or both, through the following classification:

- *emphasis on the software engineering aspect of TD* – the study exploits software engineering technologies for managing TD, and uses only basic terminology for referring to the financial aspect of technical debt (see Chapter 2.3); An example is the work of Chin et al. [P9] that quantifies the interest of TD on the basis of the number of design smells, without using any financial approaches.
- *emphasis on the financial aspect of TD* – the study adopts a financial approach for handling technical debt (see Chapter 2.3), but without an explicit relationship to software engineering technologies; An example is the work of Alzaghouli and Bahsoon [P2] that assesses the future and present value of the amount of technical debt using real options, without providing a clear mapping between the required inputs of real options and characteristics of the examined software.
- *emphasis on both software engineering and financial aspects of TD* – the study adopts a financial approach for managing technical debt (see Chapter 2.3), and provides a detailed analysis on how software engineering technologies can be used; An example is the work of Seaman et al. [P53] that suggests the management of TD using Cost-Benefit analysis, Portfolio Management and Real Options considering as inputs design smells which are present in software.
- *emphasis on neither software engineering nor financial aspects of TD* – the study is using only the basic terms for referring to the financial aspect of technical debt (see Chapter 2.3), and does not exploit software engineering technologies. An example is the article by Rooney [P47] that discusses the need for a new metaphor of technical debt claiming that the metaphor is inappropriate for systems built with modern development approaches such as Agile practices.

The results indicated that 56% of the studies that examined technical debt emphasize on the SE aspect of technical debt, without specific focus on any financial aspect (i.e. other than the basic terms discussed in Chapter 2.3). Next, in about 22% of the studies the two aspects were balanced, i.e. employed financial approaches for managing technical debt, by using some software engineering technologies. However, we note that the level of detail in one of the two parts might lag compared to the other (see discussion in Chapter 2.5.2). Finally, research efforts that mainly emphasize on the use of financial approaches are rather limited, i.e. about 10% of the primary studies, while the rest 12% of the studies can be classified as having limited emphasis on both the software engineering and the financial aspect of TD. The rest of this chapter presents the results of data collection and analysis, organized by research questions.

2.5.1 Financial terms related to technical debt management (RQ₁)

By focusing on the terms used to describe the financial aspect of TD, the results are summarized in Table 2.5.1. For each term, we list the frequency (i.e., the number of distinct studies in which the term is being used), and the corresponding percentage over the total number of primary studies. We note that one paper, might employ more than one term; in this case all relevant terms have been recorded. However, for a paper to be counted, the corresponding term should form an integral part of its discussion or methodology.

We observe that *interest* and *principal* are the most frequently used ones. Additionally, we can observe that in many cases, conceptually similar terms may not be used in the primary studies under a uniform term. For example: the terms *value* and *business value* are both used in different studies with similar meaning. We also note that some of the basic financial terms presented in Chapter 2.3 (i.e., debtor and creditor, debt instrument, maturity date, and liquidity) are not investigated in technical debt literature. This raises the question whether these terms have been deliberately left out or whether they can be applied in the technical debt domain. For example, can *technical debt have a maturity date* or *does liquidity constitute a factor on the management of technical debt*?

Table 2.5.1: Financial terms used in technical debt research

Term	Frequency	Percent	Term	Frequency	Percent
interest	29	42.03%	compound interest	2	2.90%
principal	17	24.64%	benefit	2	2.90%
cost	8	11.59%	bankruptcy	2	2.90%
repayment	7	10.14%	future value	2	2.90%
return on investment (ROI)	7	10.14%	by-product	1	1.45%
asset	5	7.25%	total cost of ownership (TCO)	1	1.45%
investment	4	5.80%	depreciation	1	1.45%
value	4	5.80%	effort	1	1.45%
risk	4	5.80%	financial leverage	1	1.45%
present value	3	4.35%	cash flow	1	1.45%
productivity	3	4.35%	hedging	1	1.45%
opportunity cost	3	4.35%	loan shark	1	1.45%
business value	3	4.35%	opportunity benefit	1	1.45%
option	3	4.35%	voice of market	1	1.45%
liability	3	4.35%	savings	1	1.45%
interest rate	2	2.90%	value-added	1	1.45%
revenue	2	2.90%	voice of business	1	1.45%
capital	2	2.90%	voice of customer	1	1.45%
net present value	2	2.90%			

Although we acknowledge the fact that some of these terms might have already been used in related research areas such as software economics (e.g., net present value of software development strategies (Erdogmus, 1999), valuation of software refactorings (Mavridis et al., 2012).), their results have not yet been highly exploited by the technical debt community. To this end, we suggest that the research community of technical debt might benefit from reusing research results from other related domains (some examples for measuring technical debt by using software economics (Boehm and Sullivan, 2000) are presented in Chapter 2.5.2).

2.5.2 Financial Approaches for Managing Technical Debt (RQ₂)

According to Li et al. (2015), technical debt management, can be decomposed to five activities: identifying, measuring, prioritizing, repaying, and monitoring TD. Since measurement forms the basis for any financial handling of TD, we separately discuss financial approaches with respect to measurement (see RQ_{2.1}), and group the discussion on all other TDM activities (see RQ_{2.2}).

Financial Approaches for Measuring Technical Debt: In the table of Appendix A.2⁷, for each financial approach related to technical debt measuring we provide: (a) a pointer to the corresponding primary studies, (b) a brief description of the used method, (c) the software engineering technologies that they use as input, (d) the measured financial terms, (e) the used financial approaches, and (f) the TDM activity that they refer to. As financial approaches we consider all the approaches that have been presented in Chapter 2.3, under *Managing Debt Strategies*. In case a study only used software engineering tools or methods to quantify technical debt (i.e. the approach is not financial), it has been classified as based on *Software Economics*. For example, a study would be characterized as using software economics for measuring technical debt if:

- it considers the number of bad smells as an estimate of accumulated technical debt, and then valueate this number in terms of money that have to be spent for resolving them; or
- it uses software effort/cost estimation methods (size, cyclomatic complexity, etc.) for assessing the effort/cost required to enhance particular system design-time qualities (e.g., maintainability) of a software component.

To provide an overview, we synthesize the data from Appendix A.2 through cross-tabulating the measured financial terms and the corresponding approaches, as presented in Table 2.5.2. Similarly to Table 2.5.1, frequency refers to the number of papers, in which the corresponding terms are measured. From Table 2.5.2, we can observe that most finan-

⁷ Table has been moved to Appendix A.2 to improve the readability of the paper.

cial terms have been quantified through software economics methods (73.1% or 19 out of 26), whereas the most frequently quantified terms are *amount of debt* (i.e. *principal + interest amount*) and *interest*.

Table 2.5.2: Measuring Technical Debt Approaches

Measured Term	Frequency	Measuring Approach			
		Portfolio Management	Real Options	Software Economics	Value Based
<i>Amount of debt</i>	9	0	1	7	1
<i>Compound Interest</i>	1	0	0	1	0
<i>Interest</i>	6	1	0	5	0
<i>Repayment</i>	1	0	0	1	0
<i>Principal</i>	4	1	0	3	0
<i>ROI</i>	1	0	0	0	1
<i>Simple Interest</i>	1	0	0	1	0
<i>Future Value</i>	1	0	1	0	0
<i>Present Value</i>	1	0	1	0	0
Total	26	2	3	19	2

Based on the inherent financial nature of TD, an approach that measures technical debt is expected to calculate the amount of technical debt (or related terms) as values in money or other similar units. By further focusing on the level of granularity of the items that these approaches receive as inputs, we can identify two basic trends:

- Approaches that receive ***coarse-grained inputs***, i.e., inputs that have to be estimated based on a number of factors. For example, in this category we classify approaches that use effort as input, in the sense that effort spent on several activities (e.g., identification of code smells and application of refactorings) is related to the size of the artifacts, the number of bad smells, the intensity of the bad smells, the urgency to solve them (based on artifacts history), etc.
- Approaches that receive ***fine-grained inputs***, i.e., inputs that can be estimated directly without taking other factors into account. For example, in this category we classify approaches that use as input the number of code smells, in the sense that the number of code smells can be measured directly from the code.

Approaches receiving coarse-grained inputs for measuring technical debt usually originate from economics (traditional or software economics). In the domain of cloud-computing, Alzaghoul and Bahsoon define the amount of debt as the *additional effort needed to provide scalability to a SOA* [P2]. On the other hand, concerning traditional software devel-

opment, de Groot et al. [P22], Nord et al. [P41], Nugroho et al. [P42], and Stochel et al. [P60] define the amount of debt as the rework cost that is needed to improve the levels of quality. Building on that, Schmid [P50] specifies that in order to quantify TD, you need to assess the effort needed to refactor a software product so as to have zero debt. In a similar context, interest is defined as the difference in the amount of technical debt. Chin et al. [P9] define simple interest as the *ongoing product maintenance cost* and compound interest as the *increase in the amount of technical debt over time*.

Concerning approaches that receive fine-grained inputs, Guo and Seaman [P23] and Guo et al. [P24] use historical data on effort, so as to estimate the effort needed for bug-fixing or increasing test coverage, as indications of technical debt. Curtis et al. [P12] count code violations and classify them based on their severity (high, moderate and low), and then calculate the total time needed for fixing these violations. Finally, Santos et al. [P49] use cohesion and coupling metrics, code duplications, lack of comments, coding rules violation, potential bugs, the absence of unit tests, etc. for assessing code technical debt.

It becomes clear that all methods used for measuring TD that are solely based on software economics, are not related to any approach related to financial debt. On the other hand, all studies that apply financial approaches to technical debt measurement appear to be more balanced. Specifically, Alzaghouli and Bahsoon [P2] apply Real Options Analysis and use a two-step binomial tree approach in order to quantify technical debt, in the case of web services substitution. Technical debt is considered to be the difference between the option's value and the cost of switching to a new web service. The value of the system, the option price, the option value, and the probability coefficient are derived through functions, using values based on historical data. However, the use of interest rate (r) in the model is not clearly defined.

In addition, Guo and Seaman [P23] measure technical debt principal and expected interest amount in person-days as the effort needed to resolve a technical debt item and the extra effort required to complete the task later, respectively. In order to fit their measurements in a Portfolio Management model, they consider TD items as assets and they also determine the likely net benefit of the item and the risk that the item will not produce benefit. The likely net benefit is considered to be the expected asset return and is defined as the principal minus the expected interest amount. On the other hand, in Portfolio Management theory, the risk is considered to be represented by the variance of return. The authors suggest that, in the case of TD, this is equal to the interest standard deviation. All estimations are based on historical effort, usage, change, and defect data.

Finally, Stochel et al. [P60] propose a three-layered model and define codebase/design debt, architectural debt and portfolio debt. They propose the SonarQube approach and the Wisdom of Crowds technique for the measurement of design debt. Regarding architecture

debt, they propose that its estimation should be based on the cost of expected changes in the architecture. At portfolio layer, technical debt is defined as the variation of the cost of change in order to achieve the highest ROI.

Financial approaches for identifying, prioritizing, repaying and monitoring technical debt: In contrast to approaches that have been introduced for measuring technical debt, in which only limited studies use financial approaches while quantifying technical debt, in the rest of the management activities, we observe that the financial aspect is more evident. In the table of Appendix A.3, we present the financial approaches that have been proposed for managing technical debt. Specifically, we provide: (a) a pointer to the corresponding primary studies, (b) a brief description of the used method, (c) the software engineering technologies that they use as input, (d) the used financial approaches, and (e) the TDM activity that they refer to. The results in Appendix A.3 are synthesized and visualized in the pie chart of Figure 2.5.2.

Figure 2.5.2: Managing Technical Debt Approaches

In addition, we also investigated the financial approaches that have been used for managing technical debt in terms of: (a) the software engineering technologies that have been used as input/output of these approaches (if applicable), and (b) the potential omissions or assumptions that researchers have done while adopting the financial approach in the technical debt domain. In the next paragraphs, we present more details about the three most popular approaches for managing technical debt: cost/benefit analysis, real options analysis and portfolio management.

Cost/Benefit Analysis: Regarding cost/benefit analysis theories that have been transformed so as to fit the management of technical debt, we were able to identify six relevant studies. Buschmann [P8] presents some industrial case studies, indirectly referring to cost/benefit analysis by presenting real-world examples on the advantages and disadvantages of repaying technical debt, without however using the original terminology implied by the financial

method, or presenting software engineering approaches to assess the cost and benefits of technical debt repayment. On the contrary, Guo et al. [P24] and Seaman et al. [P53] have employed a cost/benefit analysis in a way in which costs have been related to interest amount (calculated based on interest probability and changeability analysis) and benefit has been related to the principal. However, the amount of the principal is assumed to be known, and is not calculated in the paper. To support these claims the authors present a small size case study.

In addition, Schmid [P50] presents a theoretical framework that enables the application of cost/benefit analysis during software evolution. Initially he suggests that cost is related to development effort, to product time-to-market, and to customer benefits. However, in the rest of the paper he only focuses on development costs, by taking into account the cost of refactorings as part of iteration costs. As benefit of repaying technical debt he characterizes saved cost by technical debt item, by iteration. For calculating cost and benefit, Schmid is based on existing software cost estimation models, and clearly notes that the accuracy of his model cannot be higher than the accuracy of the used cost estimation model [P50]. Furthermore, Snipes et al. [P58] propose the use of cost/benefit analysis for making decisions on bug fixing prioritization. In this study, the authors propose a method for calculating the cost of each defect fixing activity (based on type, condition and change proneness), and the expected benefit from fixing the defect (based on severity, existence of work around, etc.). The specific application of the method is sound, since it uses the cost/benefit analysis ratio, based on well-defined parameters [P58]. Finally, Zazworka et al. [P66] implement a cost/benefit matrix in order to identify and prioritize technical debt items (identified God classes (Martin, 2003)) refactoring. The cost of each refactoring opportunity is assessed by using detection strategy metrics (e.g. Weighted Method Count – WMC, Tight Class Cohesion – TCC and Access To Foreign Data – ATFD), whereas the benefit is quantified by class change and defect proneness.

Real Options: Next, concerning studies that employ the real options theory, Alzaghouli and Bahsoon [P2] describe asset value as the quality of service (e.g. scalability and availability), and option price for leaf nodes as the difference between system value and maintenance cost. On the other hand, the probability coefficient of improvement, the expected pay-off when the value goes up, the expected pay-off when the value goes down, and the interest are not directly associated with software technologies. However, the authors state that such input parameters can be assessed by stakeholders' input, historical data, or be based on existing valuation methods. In contrast to the previously well described application of the real option theory, Power [P43] characterizes the time that development teams spent on paying off technical debt as an option and investigates its impact on team's capacity and velocity, without however defining any of the aforementioned input variables or

applying the real options theory in practice. Finally, Seaman et al. [P53] also described that paying-off technical debt (e.g. through refactorings) can be managed as an investment with short-term cost (e.g. resources, effort on applying refactorings.) that can be valued through the decrease of technical debt in the long term. However, in this short paper the major focus of the authors was not to present in detail how a financial approach could be applied, but to debate on the fitness of several financial methods and their relationship to technical debt.

Portfolio Management: With respect to studies that attempt to manage technical debt through portfolio management theory, we have been able to identify three related papers. Guo and Seaman [P23] and Seaman et al. [P53], use technical debt items as assets and populate the portfolio of the software development organization with them. In an iterative way, each item is assessed and is either preserved or excluded from the portfolio. In each iteration, the method suggests excluding from the portfolio technical debt assets that should be repaid. The decision to repay an item or not is based on risk assessment. The risk assessment is being performed based on the principal, estimated interest rate, interest standard deviation and dependencies between technical debt elements. Finally, Stochel et al. [P60] suggest that the debt portfolio is layered into process debt, architecture debt, and design/source code debt portfolio. Each one of these distinct portfolio technical debt items, should be individually assessed (e.g. for architecture technical debt by using ATAM, and for source code debt by using methods/tools like the Wisdom of Crowds).

2.5.3 Software engineering technologies used by financial approaches in TDM

Table 2.5.3 presents the frequency of software engineering technologies that have been used in financial approaches in technical debt management. We note that as software engineering technologies we collectively refer to any element in the backbone of the software development process according to RUP (Kruchten, 2000) (i.e., tools, software artifacts, development activities / phases), as well as in quality assessment / improvement process (e.g., quality attributes, quality measurements, refactoring opportunities.). Such elements can provide information to be fed as input to the applied financial approaches. For example, Schmid [P50] applied the cost/benefit analysis, to decide whether technical debt items should be paid off, considering as input the refactoring cost, the relative technical debt per refactoring and per evolution step, and also the probability that a particular evolution step will be performed. According to the aforementioned definition of software engineering technologies, in Table 2.5.3, we organize the data in the following categories: *tools*, *artifacts*, *phases/activities*, *quality attributes* and *quality assurance-related terms* (e.g., quality models). We note that terms that fit two categories (e.g., design can refer to both be *design activity* and the *design artifacts*), have been classified to both classes.

Table 2.5.3: Software engineering technologies used in technical debt research

Group	Term	#	Percent	Group	Term	#	Percent
Tools	SonarQube	6	8.51%	Phases / Activities	implementation	17	24.28%
	AIP	2	2.90%		static analysis	15	21.42%
	CodeVizard	2	2.90%		testing	11	15.71%
	FindBugs	2	2.90%		maintenance	11	15.71%
	CLIO	1	1.45%		design (*)	9	12.85%
	DebtFlag	1	1.45%		architecture (*)	6	8.51%
	Microsoft Tree Mapper (tool)	1	1.45%		agile development	6	8.51%
Artifacts	source code	17	24.28%		refactoring	6	8.51%
	design (*)	9	12.85%		requirements (*)	4	5.71%
	architecture (*)	6	8.51%		evolution analysis	2	2.90%
	requirements (*)	4	5.71%		dynamic analysis	2	2.90%
	code smells	3	4.28%		project management	2	2.90%
	design smells	3	4.28%		acceptance testing	1	1.45%
	design patterns	3	4.28%		ATAM	1	1.45%
	incomplete documentation	1	1.45%		commonality-variability analysis	1	1.45%
Quality Attributes	maintainability	11	15.71%	Quality Assurance	quality dashboard	2	2.90%
	defect-density	10	14.28%		SIG/TUV	2	2.90%
	functionality	2	2.90%		SQALE	2	2.90%
	reusability	1	1.45%		ISO/IEC 25010	1	1.45%
	performance	1	1.45%		ISO/IEC 9126	1	1.45%
(*) these terms are duplicate in the table					COCOMO	1	1.45%

2.6 Discussion

The results of the study provide useful insights towards the two goals set in Chapter 2.1, the glossary of terms and the classification of approaches. In this chapter we synthesize the abovementioned results into the two envisioned goals: a glossary of well-defined descriptions of all financial terms that have until now been connected to technical debt (see Chapter 2.6.1 – related to goal-a); a classification schema of the application of financial approaches in technical debt research (see Chapter 2.6.2 – related to goal-b).

2.6.1 Technical Debt Financial Glossary

In this section, we provide a glossary of the financial terminology that is used in technical debt research, presented in Table 2.5.1. In order to compose the glossary, we have used the terms and their meanings from the primary studies. We derived the definition for each term, by synthesizing the way that these terms are used in several studies. In cases of conflicted definitions (explicit or implied), the terms are defined based on our own perception of the way that these concepts could prove beneficial for technical debt management (see

Asset:	<ol style="list-style-type: none"> 1. The effort that the development team saves by producing immature artifacts while accumulating technical debt. 2. Tools or approaches that lead to the decrease of technical debt.
Bankruptcy:	A software engineering project could be considered bankrupt, in case that it fails to survive/evolve (either being cancelled or forced to be re-written from scratch) due to large maintenance costs, caused by the accumulation of technical debt.
By-product:	An additional software product that can be produced with the effort that has been saved from producing immature artifacts while accumulating technical debt.
Financial Leverage:	The ability of software companies to produce by-products, decrease time to market, etc. by accumulating technical debt.
Future Value:	The value that an action (e.g. undertaking technical debt) performed now will have after a given time period.
Interest:	The additional effort that is needed to be spent on maintaining the software, because of its decayed design-time quality.
Liability:	TD items, i.e. artifacts that have not been developed with the optimal design-time quality and thus subject to improvement.
Present Value:	The value that an action (e.g. a repayment activity) to be performed in the future has now.
Principal:	The effort that is required to address the difference between the current and the optimal level of design-time quality, in an immature software artifact or the complete software system (syn: capital).
Repayment:	The amount of effort spent on improving design-time quality. This effort will decrease the effort needed for future maintenance tasks.
Risk:	The probability or threat that the technical debt items are accumulated in a design hotspot. Immature artifacts in such places of the system can hinder a product's viability.
ROI:	The ratio of (1) the additional amount of money that is earned by bringing the product earlier into the market or (2) the additional amount of money that has been earned from the company by investing the effort of the principal in an activity different than the improvement of design-time quality, over the principal.
Value-Added:	The additional value that repayment of technical debt would bring to the software product.

Figure 2.6.1.a: Technical Debt Financial Glossary

Figure 2.6.1.a). We also performed a merging of terms that were similar in meaning. From the glossary of Figure 2.6.1.a, we have intentionally excluded the following terms, because their meaning in TD is completely equivalent to their meaning on financial debt: benefit, cash flow, cost, depreciation, hedging, investment, loan shark, option, opportunity cost, productivity, savings, voice of business, voice of customer, voice of market, and total cost of ownership. Finally, we note that the original (i.e., financial) definitions of terms have been omitted, although they might hold in the TD context. For example, the software product per se is an asset for the company, since the company can sell it and make value from it; however, this value is outside the context of TD.

Next, all terms of the glossary are discussed in detail, and graphically illustrated in Figures 2.6.1.b – 2.6.1.d. The intention of these figures is not to quantify the terms, but to provide a graphical context in order to facilitate their understanding. To further enhance the readability of the discussion, we group the terms, based on if they are applicable to:

- the pre-deployment phase when technical debt is accumulated – Figure 2.6.1.b;
- the post-deployment phase, considering changes related to the implementation of new requirements – Figure 2.6.1.c;
- the post-deployment phase, considering changes related to the improvement of system design-time qualities – Figure 2.6.1.d.

We note that the values in all axes of Figures 2.6.1.b – 2.6.1.d are illustrative and do not originate from any empirical data, or reference. In addition, the Figures illustrate individual artifacts as items that are subject to technical debt (see for example [P23], [P53], [P60], and [P66]); however, TDM can also be performed at system-level (e.g., see for example [P2]). The depictions and explanations are equally applicable to system and artifact level.

In Figure 2.6.1.b, we assume a software system that is composed of 7 artifacts (e.g. software components). Artifacts 2 and 3 have been developed on the desired levels of design-time quality, whereas in all other artifacts several compromises have been made ($\delta_{quality}$). The immature software artifacts are named *technical debt items* or *liabilities*. While developing the aforementioned artifacts the development team spent less effort than it was required in the optimal case resulting in $\delta_{quality}$. The effort that is required to address this difference in the levels of quality is termed *principal* (or *capital*) of the TD. Principal can be considered as an *asset* for the company, since it can be invested in any other activity due to *financial leverage*. Such activities can be the development of *by-products* or *decreased time to market*. The value of such by-products divided by the principal represents the *return of investment (ROI)* of investing the principal into other activities

Figure 2.6.1.b: Pre-deployment Phase

Figure 2.6.1.c: Post-deployment changes to implement new requirements phase

In Figure 2.6.1.c, the y-axis refers to effort instead of quality. We suppose that the same system, after its deployment, requires the addition of a feature. For simplicity, we assume that this feature is global and the same effort needs to be spent in every component (artifact). However, in the artifacts where technical debt is accumulated (technical debt items), additional effort (δ_{effort}) is required because of their deteriorated design-time quality (e.g. low maintainability, bugs that are found during extension, incomplete documentation that lead to low understandability). This additional effort is the *interest* that the development team has to pay, due to the accumulated amount of debt. If the interest becomes so high that maintenance is not financially feasible or beneficial, the project becomes *bankrupted*.

In Figure 2.6.1.d, we consider the evolution of a system with accumulated technical debt (accumulated technical debt is calculated as: *principal + interest*). The two series in the line chart represent the evolution of the system without any repayment activity (red line – inherent) and with one repayment activity performed in revision-4 (blue line – with repayment). We suppose that the system starts with an amount of debt that increases over time (interest rate is represented by the slope of each line, e.g., $\theta 1$ and $\theta 2$). As we observe, in the case of Figure 2.6.1.d, the interest rate is not *stable*, but *floating*, since the slope of the red line is increasing over time (i.e., $\theta 2 > \theta 1$ and $\varphi 2 > \varphi 1$). This increase is expected, since the design-time quality of decayed projects, deteriorates quicker than the quality of better-designed products (the poor getting poorer). Therefore, the *TD risk* for low design-time quality products is higher than the TD risk of high design-time quality products. For this particular example, we assume that the interest rate increases, not because of the introduction of new flaws or bugs, but due to the increased difficulty of resolving existing problems, which in turn is caused by the gradual increase in size and functionality. Due to the *repayment* actions in revision-4, some artifacts' design-time quality is increased, i.e. technical debt is decreased (δ_{TD_i}). The effort that is spent during this repayment activity is the *value of repayment*. Furthermore, because of the floating rate of the interest (it increases in revision-3 (TD=50) and in revision-6 (TD=75)), we can see that in revision-7 the value of repayment increases, as illustrated by the distance between the two lines ($\delta_{TD_7} > \delta_{TD_i}$). The value of the repayment performed in revision-4, is named *future value of repayment*, in the timestamp of revision-7. Assuming that present time is revision-2 and that a future repay-

Figure 2.6.1.d: Post-deployment changes to improve design-time quality

ment activity (e.g. the one performed in revision-4) should be valued at present conditions, one can assess at revision-2 the *present value of the repayment* to be performed in revision-4. Finally, according to de Groot et al. software value is, among others (e.g., volume/functionality), related to product quality (i.e., inherent characteristics of the software) [P22]. Therefore, the enhancement of design-time quality that is achieved through the repayment on revision-4 represents the *value-added* of TD repayment.

2.6.2 Classification Scheme for Financial Approaches used for Managing Technical Debt

In this section we present a classification scheme for techniques on managing technical debt (see Figure 2.6.2). The classification has been built based on the raw data of this SLR (presented on Appendices B and C). For readability reasons, while developing the classification schema we preferred not to list the primary studies that should be mapped to each edge. In any case, the interested reader can find all relevant information in Appendices B and C. According to Nickerson et al., the most common paradigm for building classification schemas for information systems is the three-level indicators model, which is based on both empirical and deductive approaches (Nickerson et al., 2009). By applying this model, we: (a) examined the objects (i.e. studies), (b) we identified general distinguishing characteristics of the objects, and (c) we grouped their characteristics so as to create our classification schema (Nickerson et al., 2009). Specifically, in step (b) we identified three *characteristics* that will constitute the three levels of the proposed schema:

- the 1st level of the schema represents the existing activities within technical debt management (Li et al., 2015) – extracted variable: [A15] – last column of Appendices B and C;
- the 2nd level represents the proposed financial techniques – extracted variables: [A11] and [A12] – 5th column of Appendix A.2 and 4th column of Appendix A.3;
- the 3rd level corresponds to the used software engineering technologies – extracted variable: [A13] – included in the 3rd column of Appendices B and C.

In addition, in the proposed schema the popularity of each technique in level 2, and of each software engineering technology in level 3, can be deduced by the number of edges reaching the corresponding node, respectively. More specifically, the lines connecting the categories of level 1 to the approaches of level 2 and the latter to the technologies of level 3 indicate which elements are connected to each other in the primary studies. In case more than one edges connect two nodes, a multiplicity symbol is used. For example, the *cost/benefit analysis* has been encountered in 3 studies discussing the *repayment* of technical debt. To improve the readability of Figure 2.6.2, edges leaving from the same financial approach (at level 2) are grouped through a common line style.

Moreover, we evaluate each technique of level 2, w.r.t. the cross-examination of software engineering technologies and financial approaches. Specifically, the number of arrows under the names of each financial approach denotes how many times it is applied, regardless of the technical debt management activity (e.g., *value-based* approaches have been applied 3 times in TD research). Approaches marked with upwards-pointing green arrows are adequately applied by focusing on both financial and software engineering aspects; approaches marked with downwards-pointing red arrows miss one aspect (either financial or software engineering); those marked with a horizontal yellow arrow present a mapping between software engineering and financial aspects, but their explanations are not detailed. For example, the *cost/benefit* analysis has been adequately applied in 4 studies, by relating financial and software engineering aspects, in 1 study both aspects are discussed but valuation of software activities are not thoroughly discussed, while in 1 study the authors have focused only on the financial aspect of technical debt. As evaluation for the proposed classification schema, we attempted to validate the four desirable attributes for successful classification schemas, as proposed by Nickerson et al. (2009). According to Nickerson et al., a successful classification should ensure:

- ***conciseness***: contain a limited number of dimensions or a limited number of characteristics in each dimension. (Nickerson et al., 2009);
- ***inclusiveness***: contain dimensions and characteristics to be of interest (Nickerson et al., 2009);
- ***comprehensiveness***: provide for classification of all current objects within the domain under consideration (Nickerson et al., 2009); and
- ***extendibility***: allow for additional dimensions and new characteristics within a dimension when new types of objects appear (Nickerson et al., 2009).

We note that conciseness can sometimes be compromised in favor of inclusiveness. Therefore, we selected a reasonable depth for our classification schema (i.e., 3) and a small amount of characteristics in the first two dimensions (i.e., 5 and 6 respectively), so as to provide a *concise* schema that will be understandable and applicable in practice. On the other hand, we preferred to include a large amount of classes in the third level, so as to be as *inclusive* as possible with respect to software engineering technologies. Additionally, the *comprehensiveness* of the proposed schema is guaranteed by the fact that we were able to classify each study in exactly one category. Finally, *extendibility* is related to the effort required to extend the schema: adding classes is an easy extension, while adding levels can be far more complicated since it can result in changing the existing mapping between classes and levels. In this sense, we believe that the proposed three levels of the schema (TDM activities, financial approaches, and software engineering technologies) are sufficiently generic to reflect a large body of research in TD without requiring additional levels. On the

other hand, we expect additional classes to be added, representing novel financial approaches and software engineering technologies that will be proposed in future research. From the first level of the schema (i.e., TDM activities) we can identify classes, i.e. research topics that need further investigation. First, we can observe that technical debt identification is not connected to any financial approach, whereas research on prioritizing is also neglected, compared to monitoring, repaying and measuring technical debt. Repaying technical debt is in most cases explored through cost/benefit analysis, which however sometimes is not adequately applied (see Chapter 2.5.3 for details). Additionally, real options, although quite popular as a technique for managing technical debt, they: (a) seem to be used without a clear mapping of financial and software engineering technologies, (b) are in many cases applied in an artificial rather than in a real-world setting. Finally, concerning the application of financial approaches on TDM, we observe that researchers do not uniformly map software engineering technologies to the inputs required by the financial methods. For example, in the cost/benefit analysis, researchers use a variety of software engineering technologies, which indicates that there are many alternatives that can be used as inputs (e.g., refactoring opportunities at the architectural level, smells at the source code level, quality metrics.).

Figure 2.6.2: Financial Approaches Classification Schema

2.6.3 Implications for Researchers and Practitioners

Based on the expected contributions (see Chapter 2.1), we believe that the reported results can prove beneficial to both researchers and practitioners. Firstly, we encourage software practitioners:

- to use the “correct” financial terms while communicating their technical debt concerns, to managers. Under this perspective, software engineers are expected to find the technical debt glossary a useful tool that will enhance communication issues;
- to use the classification for guidance on how each technique can be applied and for which management activity. For example, measuring is usually done with software engineering methods, prioritizing and monitoring with portfolio management, and repaying with cost/benefit analysis;
- to use the classification for mapping the inputs and outputs of the financial approaches to software engineering technologies. For example, the classification suggests that prioritization of technical debt items can be performed through cost/benefit analysis, applied at both source code and architecture level. The benefit can be quantified by measuring the improvement resulting by structural changes (e.g., refactoring) or defect elimination (debugging). In this example, design-time quality of the product and number of defects are mapped to the inputs of the financial approach.

Secondly, regarding researchers, the results of the study indicate that there are specific issues in technical debt research that need further attention:

- at this point the use of financial terms is rather ambiguous. Therefore, we prompt researchers to “correctly” use financial terms (as close to their original definition as possible), and clearly describe any possible deviations from them;
- technical debt research is interdisciplinary. Therefore, researchers should present both aspects (financial and software engineering) and make the connection between the two visible. In addition, although, we acknowledge that in most cases it is inevitable to focus on one aspect, the omission of the other should be discouraged;
- the application of a financial method on software is not an easy task. Most software engineers lack in-depth knowledge of financial approaches, and should spend more time on understanding the method and apply it as appropriately as possible, by clearly explaining the mapping of terms of one discipline on the terms of the other;
- there are specific financial aspects (i.e., terms) of technical debt that have not been quantified or taken into account in technical debt research (compare results of Table 2.5.1 to those of Figure 2.6.1.a). However, these terms are transferable to technical debt research, and by using them, a new range of approaches becomes applicable to technical research;

- the research domain of technical debt management is in need of a common reference point from a terminology perspective. For example, in this study we used and extended the classification of technical debt activities proposed by Li et al. (2015). However, in other studies, such references appear to lack. To this end, we present a glossary that can be used as a starting point for the construction of widely accepted list of terms, and method classification.

2.7 Threats to Validity

In this chapter, we discuss possible threats to the validity of our study. To the best of our knowledge there is no established categorization of threats to validity for secondary studies, in contrast to other types of empirical research, e.g., case studies (Runeson et al., 2012), or experiments (Wohlin et al., 2000). Therefore, we organize this chapter in four major categories: (a) threats to identification of primary studies, (b) threats to data extraction, (c) threats to generalization of results, and (d) threats to conclusions.

2.7.1 Threats to Identification of Primary Studies

Threats to the identification of primary studies deal with possible limitations of the article search process that can lead to missing related literature. In our search process, any study that does not mention the word “technical debt” in the title, abstract or keywords of the article has been excluded from the primary studies set. So, a number of articles that deal with technical debt might have been omitted. However, we believe that papers that focus on technical debt would most probably explicitly state it in their titles, abstracts or keywords. Other papers that use the term in the full text are probably just making a brief mention to it, without leading to loss of valuable data for our research.

2.7.2 Threats to Data Extraction

Threats to data extraction deal with possible problems that might arise in the data collection phase. The most common threat during this phase is the subjectivity of the researcher who performs the data collection. In order to mitigate this threat, we developed a data collection validation mechanism that involved at least two researchers, as it is presented in Chapter 2.4.5. In addition to that, especially for extracting the data for variables [A11] and [A12] (i.e., ways of measuring and managing technical debt), the data extractors were advised to try to use the classification scheme proposed by Li et al. (2015), so as to avoid the use of synonyms and ease the data analysis phase.

2.7.3 Threats to Generalization

Threats to generalization refer to threats that occur while generalizing the results from our sample to the population. In this SLR, we identified three such threats. First, the results on

the evaluation of the applicability of financial approaches on technical debt only represent the way that they are used in the primary studies that we have explored, and do not capture the applicability of the approaches in general. Second, both the classification schema and the glossary are only representing the current status of technical debt research, and therefore are prone to change when additional primary studies will be published. Third, although the discussions presented in this study (i.e., the glossary of terms, the classification schema) are heavily dependent on the raw results of this SLR (i.e., a plethora of published primary studies), we acknowledge that subjective decisions (e.g., resolution of conflicting term definitions, merging of terms) have to be validated in practice. It should be noted that since about one fourth of the primary studies originate from a particular workshop (i.e., MTD), the results might overrepresent the research of this particular community. This is a rather expected outcome, in the sense that MTD is devoted to studies on technical debt, and TD is a relatively new area of research.

2.7.2 Threats to Conclusions

Threats to conclusions validity are factors that can lead to incorrect conclusions, either by identifying incorrect relationships, or by missing existing relationships. In this category, we have been able to identify two possible threats to validity. First, we measure research intensity with the number of studies and not with the volume of research or information they provide. However, such an attempt would introduce subjective criteria into the analysis of the results. For this reason, we preferred to use an objective (directly measurable) criterion for characterizing the intensity of research. Second, we evaluate the completeness of the approach based on if the study uses sufficiently both aspects of software engineering research (i.e., financial and software engineering) and not a method for level of evidence (Alves et al., 2010) or rigor (Ivarsson and Gorschek, 2011). The reason for preferring this type of assessment in favor of evaluating level of evidence or rigor was the relevance of such an evaluation to the goals of this study, i.e., to investigate if the financial aspect of technical debt is neglected against the software engineering one.

2.8 Conclusion

This study aims at summarizing the research state of the art on technical debt management, emphasizing on the financial aspect of the metaphor. The main research questions that the systematic literature review answers are: (a) which financial terms are used in the field of technical debt, (b) which financial approaches have been used for technical debt measurement, (c) which financial approaches have been used in technical debt management, and (d) which software technologies have been applied in financial approaches of technical debt management.

As an answer to the aforementioned questions, we introduce a glossary of financial terminology and a classification schema of the financial approaches used in technical debt management. On the one hand, the glossary incorporates the most important financial terms used in technical debt research and provides definitions that can be used by the technical debt community. On the other hand, the classification schema consists of three levels; the upper one constitutes of the existing categories of technical debt management (Li et al., 2015), the middle one represents the financial techniques used, and the lower level depicts the used software engineering technologies, methods or tools. Additionally, the schema provides details on the popularity of each technique, term, method or tool, plus an evaluation of the techniques of level 2, based on their limitations when they are used in technical debt research.

By incorporating the abovementioned glossary of terms, software practitioners are expected to improve their communication with non-technical managers. Moreover, they can employ the proposed classification schema as a tool that will help them apply the suggested techniques for every TD management task. Finally, the results of our research suggest that researchers should attempt to pay further attention to issues such as the interdisciplinary nature of technical debt, the appropriate application of financial approaches to technical debt research and the possibility of applying new approaches to technical debt research, based on aspects that have not yet been quantified.

Chapter 3 – The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study

Technical Debt Management has drawn the attention of software industries during the last years, including embedded systems. However, we currently lack an overview of how practitioners from this application domain perceive technical debt. To this end, we conducted a multiple case study in the embedded systems industry, to investigate: (a) the expected lifetime of components that have TD, (b) the most frequently occurring types of TD in them, and (c) the significance of TD against run-time quality attributes. The case study was performed on seven embedded systems industries (telecommunications, printing, smart manufacturing, sensors, etc.) from five countries (Greece, Netherlands, Sweden, Austria, and Finland). The results of the case study suggest that: (a) maintainability is more seriously considered when the expected lifetime of components is larger than ten years; (b) the most frequent types of debt are test, architectural, and code debt; and (c) in embedded systems the run-time qualities are prioritized compared to design-time qualities that are usually associated with TD. The obtained results can be useful for both researchers and practitioners: the former can focus their research on the most industrially-relevant aspects of TD, whereas the latter can be informed about the most common types of TD and how to focus their TDM processes.

3.1 Motivation

Embedded Software (ES), as a type of software targeting devices that is not typically thought of as computers, is usually specialized for a particular hardware and therefore has platform-specific run-time constraints (e.g., memory usage, processing power.) (Stroustrup, 2005). In recent decades, software plays an increasingly important role in the development process of embedded products: as suggested by Rauscher and Smith some embedded companies have increased the percentage of staff devoted to developing software in the organization—as much as 80% in some software-intensive domains (Rauscher and Smith, 1995). A possible explanation for that is software’s negligible replication cost and its greater flexibility compared to hardware, which makes it easier to change. Thus, product development managers often allow for some software additions or changes late in the product development cycle to address hardware problems or to add new functionality (Rauscher and Smith,

1995), *resulting in intense maintenance activities*. An additional challenge in embedded systems is the *long lifetime expectancy*, which is normally beyond a decade (see Chapter 2.3.2), requiring the management of old systems in parallel to the design and implementation of new ones. Any decision to take advantage of novel hardware and software platforms means that companies need to manage many different configurations. Based on the above, it can be concluded that maintenance is an extremely challenging and costly activity in the ES domain.

To decrease the effort spent on maintenance, companies could invest in boosting design-time quality (e.g. maintainability) (Ampatzoglou et al., 2015a). However, in most cases, companies trade design-time qualities in favor of business qualities such as product time-to-market. The relevance of this strategy for embedded software development has already been acknowledged in the literature: embedded software development is particularly challenging in the high-end technology sector, which is characterized by shortening product lifecycles, rising market fragmentation, and rapid technological changes (Bowersox et al., 1995), (Mallick and Schroeder, 2005). As we have also stated in Chapter 1, the aforementioned compromise between design-time qualities and business qualities, leads to the accumulation of *Technical Debt*. It has been widely accepted that in an *industrial context* zero technical debt is not realistic, and probably not even desirable; the investment to reduce TD to zero would be extremely inefficient (Eisenberg, 2013). Thus, establishing Technical Debt Management methods, tools and techniques are necessary to guide the (partial) technical debt repayment, the prioritization of refactoring opportunities, and the provision of design decisions that will improve the quality of software.

Although current methods, tools and techniques for monitoring and managing technical debt are continuously improving, they do not target specific application domains, like Embedded Systems. In particular, we expect that ES are in need of domain-specific methods, due to the special requirements of such software applications, e.g., hard software constraints, the need to guarantee run-time qualities. A first step toward the development of domain-specific methods, tools and techniques is the in-depth understanding of TD in the domain under consideration. To this end, in this chapter we perform an *exploratory case study* that aims at understanding *how TD is perceived* in the ES industries. This understanding can support steering research in technical debt management as well as raising awareness among ES practitioners on technical debt-related issues.

The rest of the chapter is organized as follows: In Chapter 2.2, we present related work on studies that report on TDM in an industrial context and studies that report on the important quality attributes for embedded software systems development. In Chapter 2.3, we report on the case study design, whereas in Chapter 2.4 we present our results, organized by re-

search question and accompanied with possible interpretations. The results are discussed in Chapter 2.5, with respect to the implications to researchers and practitioners that they provide. Finally, in Chapter 2.6 we report threats to validity and mitigation actions.

3.2 Related Work

We have identified a few studies on investigating the accumulation and management of technical debt in industry. Ernst et al. (2015) conducted a survey to understand which Technical Debt type is more frequent among practitioners. The results show that architectural debt is the one that is mostly mentioned by practitioners. Although 31% of the respondents belong to the embedded software domain, the study does not distinguish results with respect to the application domain. Ernst et al. have also found that the main challenges so far are the lack of systematic practices, and the inadequateness of the tools (too complex to install or to interpret and too many false positives), while another major issue is the difficulty of increasing the awareness of TD for stakeholders without software engineering expertise; the latter is also recognized by both Lim et al. (2012) and Martini et al. (2015). According to Lim et al., (2012) measuring TD is difficult for practitioners, because the impact is not uniform. Codabux et al. (2013) mention that the repayment of TD is highly influenced by the customer needs and they also discuss the fact that practitioners decide to take on debt in order to achieve their short-term objectives, although they cannot be fully aware of the potential serious long-term consequences. Martini and Bosch (2015) highlight how TD repayment can be prioritized in an industrial context, by discussing with practitioners. The major difference of these studies, compared to ours is that we focus our investigation on the embedded software domain.

In addition to studies that focus on TD in industry, as related work, we discuss papers that provide insights on the important quality attributes (QAs) in the embedded systems domain. This kind of work is pertinent as TD represents in its core a trade-off between maintainability and other qualities. Concerning the interplay between QAs in the domain of embedded systems, Del Rosso presents an architectural approach for improving the performance of software products, derived from a product family for real-time embedded systems, and its possible implications to maintainability (Del Rosso, 2008). The results suggest that by analyzing the commonalities and differences among derived products, one can extract bottlenecks and problems in the core architecture (e.g., a God class). In a similar context, Oliveira et al. investigate the relationship between non-critical quality attributes, measured by metrics obtained from source code, and performance, measured by physical metrics (i.e., memory, time, and energy) obtained from run-time monitoring (Oliveira et al., 2008). The results indicate the existence of trade-offs between quality and physical metrics, as well as the fact that quality metrics can provide information regarding high-level quality

attributes, guiding the design solution selection at early stages, which might lead to significant gain in physical characteristics later on. Finally, Feitosa et al. (2015) analyze the difference in the trade-offs among quality attributes in critical and non-critical embedded software. The quality attributes of interest were: correctness, performance, security, reusability, understandability, functionality, extendibility, effectiveness, and flexibility. Based on the results of this study, the authors provide some hints that understandability has an inverse relationship with all the other qualities. Compared to the aforementioned studies, we emphasize on (the lack of) maintainability, which can be considered a factor for the accumulated TD.

3.3 Empirical Methodology

To explore the perception of industrial practitioners on technical debt, we have performed an exploratory case study on seven embedded systems companies (see Chapter 2.3.2). The main reason for conducting this case study is that domain-specific TDM requires a detailed understanding of how the TD phenomenon is perceived in software-intensive industries. In this chapter we describe the case study design, according to the guidelines proposed by Runeson et al. (2012).

3.3.1 Objective and Research Questions

The goal of this study, described using the Goal Question Metric (GQM) formulation (Basili et al., 1994), is: “to *analyze* the perception of technical debt in the embedded systems industry *for the purpose of* understanding *with respect to*: (a) the expected lifetime of components that have TD, (b) the types of technical debt that are frequently occurring, and (c) the significance of other quality attributes *from the point of view of* software engineers, *in the context of* embedded software development”.

By taking into account the inherent relationship of maintainability and technical debt (Kruchten et al., 2012) in this study we consider maintainability as a proxy for TD. We have chosen this proxy, since the term TD is not interpreted unambiguously in industry: practitioners may mean different things when discussing TD and related concepts like Principal and Interest. On the other hand, maintainability, though not associated to a universally accepted definition, is widely accepted as the ability to make changes in a system. Moreover, even when a practitioner is not aware of incurred TD, any decisions aimed at enhancing maintainability, will result in a lower amount of TD. On the other end, if a development team is not interested in producing a maintainable system, then it is highly probable that shortcuts will be made, leading to the accumulation of technical debt. Thus, we have used maintainability as the proxy for RQ1 and RQ3; this proxy is discussed further in the threats

to validity (see Chapter 2.4). Based on the abovementioned goal, we have extracted three research questions (RQs):

RQ₁: What is the relationship between the expected lifetime of components and technical debt?

This research question is considered important, in the sense that managing technical debt is important for components, whose accumulated Interest, at some point becomes larger than the principal (Ampatzoglou et al., 2015a), (Chatzigeorgiou et al., 2015). In other words, technical debt that is accumulated on components that are not maintained for a large time period is less probable to be harmful than for components with a high expected lifetime. This aspect becomes even more important for embedded systems, since their expected lifetime is usually long. For this research question we will study the relationship between the focus of development teams on maintainability (as a proxy for TD) and the expected lifetime of the software. In other words, we will explore if software engineers focus on maintainability (potentially managing TD) when the system under development is expected to be active for many years.

RQ₂: What types of technical debt (e.g., code, architectural) are more frequently occurring in embedded systems?

According to Li et al. (2015), technical debt can be identified in all phases of software development. In particular, Li et al. have identified 10 types of TD (see Chapter 2.3.3 for more details). Each one of these TD types requires different technical debt management approaches. Therefore, understanding which the most frequent ones are can support prioritization and monitoring of TD.

RQ₃: What is the significance of building maintainable software systems (with low TD) compared to satisfying other quality attributes?

In embedded system industries, run-time constraints are very important for software development. Therefore, trade-offs between run-time and design-time qualities are a common practice (Feitosa et al., 2015). In this research question, we examine which quality attributes are taken into account in embedded software development, and how these are prioritized. In particular we aim at comparing the importance given to maintainability (the QA that is used as a proxy for TD) against the other quality attributes. By answering this research question, we gain insight into which quality attributes are prioritized in software development, and which ones are negotiable in the ES domain.

3.3.2 Case Selection

Cases and Units of Analysis. According to Runeson et al. (2012), case studies can be characterized either as holistic or embedded, based on the way cases and units of analysis are

defined. This study is an embedded multiple case study, because we investigate multiple units of analysis (i.e., components in which technical debt has been identified) extracted from multiple embedded systems industries (i.e., cases).

Case Selection. In this study, we consider as cases embedded systems companies, specializing in different application domains, and located in different European countries. A brief description of the companies is presented below and is summarized in Table 3.3.2. The companies have been anonymized due to confidentiality reasons.

Table 3.3.2: Explored Embedded Industries

ID	Company Description			
	<i>Application Domain</i>	<i>Country</i>	<i>Type</i>	<i>#Analyzed Components</i>
C1	Telecommunications	Sweden	Large	1
C2	Automotive	Sweden	Large	1
C3	Mobile	Greece	SME	7
C4	Sensors	Greece	SME	3
C5	Printing	Netherlands	Large	1
C6	Smart Manufacturing	Austria	Large	6
C7	Media Devices	Finland	SME	1

From the data of Table 3.3.2, we can observe that in our dataset we have selected a balanced mix of Small/Medium Enterprises (SMEs) and Large Enterprises. The last column of Table 3.3.2, presents the number of units of analysis (i.e., analyzed components) that have been provided by each company. In total, 20 units of analysis have been extracted and analyzed. The components from each company and some challenges and interests of the companies are briefly described as follows:

- **Company C1** contributed a telecom component that has been in use for 50 releases and is expected to be maintained for approximately 30 more years. For C1, compatibility, functional suitability, and reliability are having the highest priority so far. The major challenges that the company faces while repaying TD are: (a) the prioritization of feature development before repaying debt, and (b) the difficulty in smoothly planning refactorings. Developers are mostly interested in test and architecture TD.
- **Company C2** contributed an automotive component that has already been released 30 times from 2013, and therefore is in need of intense maintenance activities. The expected lifespan for this component is 15 more years. While developing and maintaining this component, C2 is interested in ensuring functional suitability, reliability, and

security. The company is interested in working on feature coverage test, pass rates of test cases, and prioritization of features during development.

- **Company C3** contributed seven mobile components, which are of various levels of maturity (1-21 releases so far), which have an infinite expected/desired lifespan (i.e., as far as the customer wants to use them). Although maintainability is of great interest to the company, since their embedded products might need to survive for a long period, so far, the company is mostly focused on other quality attributes (i.e., usability, functional suitability, and performance). The form of TD that the company wants to repay is within the software architecture, source code, testing, documentation, and underlying infrastructure.
- **Company C4** contributed three components (i.e., “Monitoring and processing”, “Sensor Configurator”, and “Reporting tool”). The system is a rather “young” project, i.e., half-year, and is expected to be used from the company for 5 more years. Although maintainability is characterized as important from the company, other quality attributes, like security, functional suitability and reliability are getting more priority. The company is interested in managing architectural, source code, test, and build TD.
- **Company C5** contributed the “Printing Data Path (PDM)” component, which has been used from the company for almost ten years, and its expected lifetime is 5 additional years. PDM is a highly reusable component that needs to be adapted to different printers’ hardware. Compared to the previous years, C5 is interested in putting extra effort in repaying technical debt related to testing, documentation, and design. At the same time, the functional suitability, reliability and performance of PDM are not negotiable.
- **Company C6** contributed the “Product Status Sensor”, the “Weather Sensing”, the “Product Data”, and the “Data Aggregator” components, and two components of a “Smart Product Maintenance” application. The expected lifetime of the aforementioned components is between 10 – 20 years, so their easy extension and maintenance is crucial to the company. Based on the current knowledge on the quality key-drivers for these components, functional suitability, reliability and performance should be kept at the highest level while managing technical debt.
- **Company C7** contributed a software component used for home media server applications. The component includes a platform and a Software Development Kit for application development. The software has been developed since early 2000 and during the years it has evolved significantly. The expected lifespan of the project is 20 years from now. Since the area of home automation is new and there are many players in the market, the system should be compatible with a huge number of external systems and different installations connected to different sets of home automation systems (high port-

ability and compatibility are required). Rapid implementation of the support for these systems may create technical debt.

3.3.3 Data Collection

Every company (case) has been asked to provide us with a number of components that have accumulated technical debt and are difficult to maintain. Next, for each unit of analysis the following information has been recorded:

Project Demographics

- [V1] Textual *description of project*, in which the TDI has been identified.
- [V2] First *release date* of the project. This variable aims at expressing how old the component is.
- [V3] *Number of releases* until now. This variable provides an indication on how frequent the maintenance of this component is.

Technical Debt Item Data Points

- [V4] *Estimated lifespan* of the TDI. This variable represents how significant it is to eliminate the TD from the component, since it denotes the time period during which the company is paying Interest. Due to the low number of units of analysis, we preferred to recode this variable in a categorical one, so that each data class has more members. Therefore, we recode values less than 10 years to “short-term projects” and values between 10 and 30 years to “long-term projects”. The selection of the threshold of 10 years was made since it is the median of our sample, and a decade is considered as a milestone in terms of time periods.
- [V5] *Types of debt identified*. In this variable we record what types of TD are identified in the selected component. The types have been extracted from the work of Li et al. (2015), summarized as follows: Requirements, Architecture, Design, Code, Test, Build, Documentation, Infrastructure, and Versioning. We note that from the list provided by Li et al. (2015), we have removed Defect Debt, since the TD community does not consider the existence of defects as TD principal (Kruchten et al., 2012).
- [V6] *Importance of quality attributes* along TDI evolution. A rating, in a scale from 1(lowest) to 5 (highest), of the importance of the following QAs: Functional suitability, Reliability, Performance, Usability, Security, Compatibility, Maintainability, and Portability. The quality attributes have been selected so as to represent both design-time and run-time quality attributes.

The data collection has been performed through interviews with software engineers (i.e., designers, architects, project managers) during which data have been recorded by the researchers (a supervised questionnaire-based approach (Kitchenham and Pfleeger, 2002)).

The use of supervised data collection methods ensures the understanding of the data collection instrument from the interviewees and strengthens validity. Nevertheless, the use of a supervised data collection method has restricted the size of our dataset to 20 units of analysis (obtained by 7 companies), since it was time consuming. We note that variables [V4] – [V6] have been answered by practitioners without the use of any documentation, since: [V4] represents an estimation of the respondent, in none of the companies the technical debt items ([V5]) were documented, and [V6] represents the importance of several QAs according to experts’ opinion.

3.3.4 Data Analysis

To answer the research questions set in Chapter 2.3.1, we have performed descriptive statistical analysis and hypothesis testing. The analysis plan per research question is presented in Table 3.3.4. Specifically, concerning RQ₁, we used descriptive statistics on the expected lifespan of components that carry technical debt. In addition, we explored the correlation between expected life-time and maintainability tasks to unveil a possible relationship, by performing a chi-square test. Regarding RQ₂, we report frequencies for the different types of technical debt of the highest level, as reported by Li et al. (2015). Furthermore, we present details on how industries perceive technical debt types, by synthesizing the obtained results.

Finally, concerning RQ₃, we analyzed the data in three ways: (a) we present descriptive statistics on the importance of each quality attribute separately; (b) we inspected if there are differences among the importance of quality attributes; and (c) we performed Wilcoxon Ranks Test between QAs and maintainability (the attribute that is considered as a proxy to technical debt). Using the above, have been able to spot differences in the way different quality attributes are prioritized. Such differences can lead to intentional or unintentional trade-offs. In particular, QAs that receive higher priority than maintainability are expected to be non-negotiable while managing TD.

Table 3.3.4: Data Analysis Plan

RQ	Analysis Plan	
	<i>Used Variables</i>	<i>Analysis</i>
1	V4 (Estimated Lifespan) V6 (Importance of Maintainability)	Descriptive statistics Cross-Tabulation chi-square test
2	V5 (Types of TD)	Descriptive statistics
3	V6 (Importance of QAs)	Descriptive statistics Wilcoxon Signed Rank

3.4 Results

In this chapter we present the results organized by research question. In each chapter, we first answer the questions using statistical analysis, and then provide a possible interpretation of the obtained results.

3.4.1 RQ₁: Technical Debt and ES Expected Lifetime

To answer RQ₁ we have performed cross tabulation to examine the relationship between the projects' lifetime expected duration and the attention given by practitioners to maintainability. The dataset consists of components with expected lifetime that varies from less than one to 30 years. The mean expected lifetime of the components is 12.40 years, with a standard deviation of 8.94, and a median of 10 years. As mentioned above (Chapter 2.3.3), we have added a categorical variable that separates the dataset into two groups. The first one contains components that are expected to last for less than ten years (short-term projects), and the second one consists of components with an estimated lifecycle between 10 and 30 years (long-term projects). The analysis suggests that 45% of the components belong to the short-term projects' category, while 55% of the components belong to the long-term projects one.

The results of the cross tabulation suggest that 66.7% (6 out of 9) of the components with long estimated lifecycle receive high attention in terms of maintainability (scores 'high' and 'very high'). On the other hand, practitioners do not consider maintainability issues important for the majority of the short-term components, as 54.6% (6 out of 11) of them have low maintainability importance scores ('very low' or 'low'). The results of the cross tabulation are presented in Table 3.4.1. Each column of the table represents the level of importance that practitioners have assigned to maintainability, whereas each line counts the observed and the expected number of long-term or short-term components that correspond to each level. The observed counts represent the frequencies based on experts' opinion, whereas the expected counts, represents the frequencies that would be expected based on data distribution. The closer the observed and expected counts are, the more unrelated the two variables are (Field, 2013), in the sense that the frequencies (levels of maintainability) follow the expected distribution, regardless of the values of the grouping variable (expected lifespan of the project).

Table 3.4.1: Estimated Relation Between Lifespan and Maintainability

Estimated Lifespan		Maintainability				
		very low	Low	neutral	high	very high
Long	Observed Count	0,0	0,0	3,0	5,0	1,0
	Expected Count	1,4	1,4	1,4	4,5	0,5

Short	Observed Count	3,0	3,0	0,0	5,0	0,0
	Expected Count	1,7	1,7	1,7	5,5	0,6

To objectively assess this relationship, we have performed a Fisher's Exact Test for Count Data. The test revealed that there is a statistically significant relationship ($p\text{-value} = 0.02$). The aforementioned relationship between the expected lifetime of the project and the importance of maintainability is intuitive, since software engineers are not expected to put extra effort (representing the principal of technical debt (Ampatzoglou et al., 2015a)) on the development of applications that will not be maintained for long periods. This observation suggests that the maintainability score is not independent from the expected lifespan of the system.

The technical debt management activities (TD repayment, prevention, etc.) are expected to be more relevant for projects for which long-term maintenance periods are anticipated.

3.4.2 RQ₂: Types of Technical Debt

In order to detect the most frequently identified types of technical debt in embedded systems industries, we have analyzed the data on the types of TD reported by practitioners and we have performed descriptive statistical analysis. Figure 3.4.2 presents a bar chart on the frequency of types of technical debt on the investigated industrial components (as a percentage).

Figure 3.4.2: Types of Debt Frequencies

As shown in Figure 3.4.2, the most frequently identified TD types are test debt (e.g., limited number of unit tests, lack of test automation) which is identified in 90% of the components. It is followed by code debt (e.g., duplicate code, long methods) and architectural debt (e.g., anti-patterns, best practice violations,), reported in 80% and 70% of the cases respectively. Next, documentation (e.g., outdated documentation), design (e.g., grime, de-

sign principles violation) and infrastructure (e.g., old technology in use) debt are found in many cases (55-60%). Requirements (e.g., over-engineering), build (e.g., manual build process) and versioning (e.g., multi-version support) debt seem to be the least commonly identified types of technical debt. This fact suggests that the most “tangible” and easily detectable types of technical debt (e.g., duplicate code, test case, and design artifacts) are more easily understood and reported by practitioners.

A more fine-grained analysis of the most frequent types of technical debt is presented in Table 3.4.2. The terms presented in Table 3.4.2 have been reused from the TD types reported by Li et al. (2015). In addition, from the results of Table 3.4.2, we can observe that two different subcategories of types of TD exist: (a) the types of debt that exist in the system (e.g., duplicate code) and therefore is more easily quantifiable and its potential impact can be estimated, and (b) the types of debt that refer to what is missing from the system (e.g., lack of test cases) for which only estimations can be performed.

Table 3.4.2: Frequent Types of Technical Debt

Type of Technical Debt Item	Frequency
Duplicate code	6
Limited number of unit tests	6
Complex code	3
Old technology in use	3
Lack of automated deployment	3
Violations of good architectural practices	2
Lack of test automation	2

The aforementioned results are in accordance to those of Alves et al. (2016) who are reporting on the most frequently studied types of technical debt in the research community. In particular, according to Alves et al. design, architecture, and documentation debt are the most frequently studied types in the TD literature. The fact that our results are in accordance with those of Alves et al. suggests that until now the research directions are quite accurately focused on the industrial needs. The only possible exceptions are design debt, which does not seem to be as important for practitioners as it is for researchers, and test and code debt, which does not appear to be appealing to researchers. Apart from industrial relevance, an additional parameter that makes research on these topics popular is the existence of tools for testing, code, and design quality assurance.

The most recurring types of technical debt in industry are test, architectural design, and source code debt. Architectural and design debt are among the most frequently studied by

researchers, as well. On the other hand, some types of TD (e.g., test, code, and infrastructure), which are interesting for practitioners, are understudied by the research community.

3.4.3 RQ3: Technical Debt and Quality Attributes

To investigate the relationship between quality attributes (other than maintainability) and TD, we compare the importance that is assigned to them by practitioners. The importance of each QA is presented in Figure 3.4.3. A quality attribute is considered more important if the most frequent answers are concentrated in ‘high’ and ‘very high’ scales. For example, functional suitability is considered of great importance since there was no component for which its importance was characterized as ‘neutral’, ‘low’, or ‘very low’. On the other hand, security and portability have been characterized as ‘low’ or ‘very low’ importance for approximately 50-55% of the investigated components. However, this probably reflects the choice of the companies and cannot be generalized. In particular, security is crucial for some companies in the embedded systems domain (e.g., automotive).

Figure 3.4.3: Importance of Quality Attributes

We performed Wilcoxon Signed Rank tests (De Winter and Dodou, 2010) to investigate the frequency with which each QA is given more / less priority than maintainability (which is considered as a proxy for technical debt). The results are presented in Table 3.4.3. In particular, the first column of the table represents the QAs, the second and third column depict if the quality attribute under discussion is getting higher priority against maintainability or not, and the last column stands for the p-value for the test (values lower than 0.05

denote a statistically significant difference). The QAs that receive higher importance/priority against maintainability ($p < 0.05$) are denoted with light grey cell shading.

Table 3.4.3: Priority of QAs vs TD

Quality Attribute	Most Prioritized	N	Sig.
Functionality	Functionality	19	0,000
	Maintainability	0	
	Ties	1	
Reliability	Reliability	16	0,001
	Maintainability	2	
	Ties	2	
Performance	Performance	9	0,014
	Maintainability	2	
	Ties	9	
Usability	Usability	11	0,738
	Maintainability	6	
	Ties	3	
Security	Security	4	0,125
	Maintainability	8	
	Ties	8	
Compatibility	Compatibility	2	0,705
	Maintainability	2	
	Ties	5	
Portability	Portability	3	0,104
	Maintainability	9	
	Ties	8	

From the results of Table 3.4.3 we can observe that *functionality*, *reliability*, and *performance* are given more importance than maintainability along ES development. Similarly, usability is also usually prioritized against maintainability, but this result is not statistically significant. On the other hand, for the selected systems *security*, *compatibility*, and *portability* are not the usual quality key drivers. The importance of performance in embedded systems can be justified by the limited resources (e.g., memory, execution time, energy efficiency) that are usually available in embedded systems (e.g., printers, sensors, telecommunication devices). Additionally, reliability is important for special types of embedded software, critical embedded systems (e.g., automotive, telecommunications). Finally,

the functional suitability of the system cannot be negotiated, since a system that does not conform to all of its requirements, often dictated by standards and regulations, is not ready for reaching the market. The importance of functionality, reliability, and performance, for Critical Embedded Systems is also discussed by Feitosa et al. (2015).

Regarding the rest of the QAs, security does not seem to be the first priority of software developers, since none of the studied application domains is vulnerable to external attacks (e.g., through the internet). Furthermore, compatibility and portability seem to be important only for mobile applications, due to the needs for responsive user interfaces (i.e., for devices with different screen sizes—tablets, mobile phones, laptop, etc.) and different operating systems (e.g., Android, iPhones).

While managing technical debt in embedded software, some run-time quality attributes are given higher priority than maintainability. Specifically, the ES domain prioritizes reliability, functionality, and performance against maintainability.

3.5. Implications to researchers and practitioners

The obtained results are expected to be useful for both researchers and practitioners, since the former can focus their research on the most industrially-relevant aspects of technical debt, whereas the latter can increase their self-awareness on technical debt and target the technical debt management actions inside their companies. In particular, regarding researchers, we suggest that:

- Technical debt research is in need of more methods on the test, code and infrastructure level, since these fields are identified as problematic from the practitioners, but are understudied in the literature (Alves et al., 2016).
- Technical debt prioritization (Li et al., 2015) methods should consider not only the urgency of resolving technical debt items, but also the feature prioritization of the company (see, e.g., the backlog in (Kruchten et al., 2012)).
- The development of tools and methods for TD prevention (Li et al., 2015) is necessary, so that practitioners can timely manage the accumulated TD, while developing the software.
- The development of domain-specific tools and methods that take into account the specific requirements of given fields (e.g., run-time constraint or quality trade-offs for embedded systems) and can analyze other languages that are application domain specific (e.g., php for web, or JavaScript for mobile development).

On the other hand, the results suggest that practitioners are acknowledging the significance of TD management, especially for long-term projects. On the contrary, if there is no intense

maintenance, the accumulated Interest may be lower than the principal (see (Ampatzoglou et al., 2016b) and (Chatzigeorgiou et al., 2015)). In such cases, having accumulated TD is not necessarily harmful. Additionally, the results of this study (in particular RQ₂) can provide guidance to practitioners on which TD types are the most common in the embedded systems domain (see Table 3.4.2). Based on their frequency, practitioners should be vigilant and monitor effectively the corresponding TD types. Finally, based on the importance of specific run-time QAs (see Table 3.4.3) for embedded systems, we advise practitioners to carefully select TD repayment methods, favoring those that do not harm these run-time qualities.

3.6 Threats to Validity

In this chapter, we discuss the threats to validity of this study. First, since this case study has been performed on 20 components from 7 embedded system industries, the findings and conclusions are subject to external validity threats, in the sense that using a different sample might reveal a different perception of technical debt among embedded software developers. However, the fact that the analyzed industries have different sizes and application domains provides a certain level of confidence on the obtained landscape of technical debt in ES.

The first research question on the relation between the expected lifetime and TD might suffer from construct validity threats, as the acknowledgment of maintainability as an important factor for long-lived products does not necessarily imply attention to TDM. The same holds for the third research question investigating the prioritization of TD against quality attributes; since maintainability is well-understood, but is at best only a proxy for TD, one cannot claim that the trade-off between maintainability and run-time quality attributes is directly transferrable to the notion of technical debt. Nevertheless, we consider this choice as justified since: (a) the direct use of the term TD might cause misinterpretation among practitioners (leading to a different threat to validity) and (b) the relation of TD and maintainability is established in the community (Kruchten et al., 2012). Finally, an additional threat to construct validity is the possible misinterpretation on the quality attributes by practitioners while filling in the questionnaires. However, we believe that this threat is mitigated since: (a) the data extraction method was a supervised questionnaire-based one, and therefore any possible misinterpretation has been handled by the researcher who was conducting any interview, and (b) we have chosen well established and understandable QAs.

3.7 Conclusion

The technical debt metaphor can prove to be a suitable means for conveying the importance of maintainability to the developers of rapidly evolving software domains, such as the embedded systems. Before any targeted research activities are undertaken to support the management of TD, it is important to understand the perception of the TD concept among ES developers. To this end, in this study, we have performed a case study involving seven ES industries to investigate the most frequently occurring types of TD, the quality attributes usually associated with TD items, and the relation between the expected lifetime of a component and the acknowledgement of the importance of maintainability. The findings indicate that the most recurring types of technical debt in embedded software industry are test, architectural, and code debt. At the same time, some quality attributes such as functionality, reliability, and performance are given higher priority compared to managing technical debt. Finally, developers clearly acknowledge the need for low technical debt on components that are expected to have a longer lifetime, compared to more short-lived ones.

Chapter 4 – A Financial Approach for Managing Interest in Technical Debt

Technical debt (TD) is used by both technical and management stakeholders to acknowledge and discuss issues related to compromised design-time qualities. Until now, despite the inherent relevance of technical debt to economics, the TD community has not sufficiently exploited economic methods/models. In this study we present a framework for managing interest in technical debt, founded on top of Liquidity Preference, a well-known economics theory. To tailor this theory to fit the TD context, we exploit the synthesized knowledge as presented in two recent studies. Specifically, in our framework, we discuss aspects related to technical debt interest, such as: types of TD interest, TD interest characteristics, and a proposed TD interest theory. Finally, to boost the amount of empirical studies in TD research, we propose several tentative research designs that could be used for exploring the notion of interest in technical debt practice.

4.1 Motivation

Based on the findings of the previous chapter, as well as two recent secondary studies (Ampatzoglou et al., 2015a), (Li et al., 2015), the two most frequently used financial terms in TD research are: interest and principal. Principal is a well-defined and accepted concept, which is characterized as the effort required to address the difference between the current and the optimal level of design-time quality, in an immature software artifact or the complete software system (Ampatzoglou et al., 2015a). Therefore, it is quantifiable as long as one agrees on the concerned properties (i.e., specific qualities). On the other hand, interest (associated with many definitions, which in some cases are controversial) cannot be measured in a straightforward way, since it involves the valuation of future maintenance activities. Measuring interest becomes even more complicated since its occurrence is not certain, in the sense that maintenance activities might not take place, and therefore interest will not need to be paid off. To partially alleviate these problems, in this study we conduct a systematic literature review which results in a Framework for managing Interest in Technical Debt (FITTED). The goals of FITTED are:

- (G1)** to identify types of TD interest, and the parameters that affect its valuation. Identifying the types of interest, which can occur along evolution, is the first step

towards more formal Technical Debt Management. Until now, the definitions of interest are rather high-level, and interest measurement is often not applied in practice.

- (G2) to explore how various characteristics of interest in economics apply in TD interest. An example of such a characteristic is whether interest is simple or compound. However, these characteristics have not been fully exploited in re-search state-of-the-art, yet. Therefore, in this study we first discuss the different opinions stated in the literature, and take them into account when presenting our proposal.
- (G3) to propose a TD interest theory. Until now, no study has used the economic interest theories for modelling technical debt interest. We will rely on the Liquidity Preference Theory, for modelling the evolution of TD.

The application of FITTED is expected to support software engineers to determine the change of technical debt amount in the future, by describing the parameters that affect its future value (i.e., anticipated maintenance effort, interest, etc.). This can in turn allow the use of elaborate financial methods in different TDM activities, i.e., repayment, monitoring, and prioritization. Additionally, we expect that FITTED can support further research in the field of TD, in the sense that it can facilitate a common understanding on TD interest and point to interesting research directions.

The rest of the chapter is organized as follows: In Chapter 4.2, we discuss related work, i.e., the secondary studies from which we gathered our data (see Chapter 4.2.1), and background information on interest theory from economics (see Chapter 4.2.2). Next, in Chapter 4.3, we present the research method used in this study. In Chapter 4.4, we introduce the proposed framework for managing interest in TD. In Chapter 4.5, we discuss possible paths for further research in the field of TD by employing the proposed framework, whereas in Chapter 4.6 an illustrative example of its application.

4.2 Background Information

In this chapter, we present research efforts that are related to this study, and on which the development of the proposed framework is built upon. In particular, we discuss *existing economic interest theories*. We intend to apply an existing economic interest theory, i.e., Liquidity Preference Theory, to reuse existing knowledge from economics, on how interest should be handled, and learn from accumulated experiences.

Regarding the way interest rate is defined in the market; various models have been suggested, by different schools of economics (Mishkin and Eakins, 2012). The mainstream theories are the *Loanable Funds Theory*, developed by the neoclassical school, and the *Liquidity Preference Theory*, proposed by the Keynesian theory (Mishkin and Eakins, 2012). Interest rate is the price paid for borrowing money or vice versa (the payment

received to loan money). Therefore, it can be considered as the price of money. Interest rate, as any other price, can be defined in the market at the equilibrium between supply and demand. According to the *Loanable Funds Theory*, interest rate specification takes place in the market of loanable funds. On the one hand, individuals or enterprises, who want to invest, form the demand for loanable funds. They ask for loans in order to proceed with an investment. As interest rate gets higher, borrowing becomes more expensive. As a result, demand for loanable funds decreases as interest rate increases. On the other hand, the supply of loanable funds comes from people or enterprises that use the loanable funds market to save their money. Instead of consuming part of their income, they choose to put it into the loanable funds market in order to save it for later. In this case, higher interest rate means higher return on savings. Therefore, supply of loanable funds rises as interest rate increases.

Figure 4.2.2.a: Loanable Funds Theory

In the diagram of Figure 4.2.2.a, the equilibrium in loanable funds market is presented. We note that, in economic theory, all kinds of supply – demand diagrams represent the dependent variable on the horizontal axis and the independent variable on the vertical axis. Therefore, in this case, the vertical axis depicts interest rate (r), while the horizontal axis represents the quantities of supply and demand for loanable funds. The quantity of loanable funds supplied at any level of interest rate is presented by line S . Line S depicts the positive correlation between interest rate and loanable funds supply. Likewise, the quantity of loanable funds demanded at any level of interest rate is presented by line I . The negative correlation between interest rate and loanable funds demand is indicated by the negative slope of line I . When interest rate is higher than r^* , then it is more profitable to save, or it is more profitable to lend than to borrow, and supply of loanable funds is higher than demand. On the other hand, when interest rate r is lower than the level of r^* , then it is more profitable to invest, or it is more profitable to borrow than to lend, and demand for loanable

funds is higher than supply. When $r=r^*$, then both the investors and the savers have no motivation to change their position in the market and equilibrium is achieved. Consequently, interest rate is determined at $r=r^*$. *Equilibrium* in the market is achieved at interest rate r^* , when every other factor, that could influence savings or investment, is considered stable (*ceteris paribus*). However, interest rate level may move upwards or downwards in case of changes to savings or investments, due to exogenous factors (e.g., income). For example, an increase in income would cause an increase in the quantity of savings. That would result in a shift to the right of the savings curve (S), which is the supply of loanable funds. In Figure 4.2.2.a, the new line S_1 depicts such a change. As shown in the diagram, the new equilibrium is now achieved at point E_1 and interest rate is defined at r_1 , lower than r^* .

The *Liquidity Preference Theory* determines interest rate level through the mechanism of supply and demand for money (cash), which is performed in the money market. In this case, supply of money (M) is a constant at any point of time and is determined by the central bank, according to the needs of the economy. In other words, supply of money is not dependent on interest rate and it is exogenously defined. On the other side, demand for money (L) represents the quantity of cash that people prefer to hold for purposes of transactions, precaution or speculation. In this case, as interest rate increases, it becomes more profitable for people to invest money than to hold it. Consequently, an increase in interest rate leads to a decrease in the quantity of money demanded in the market and a decrease in interest rate causes an increase in demand for money. Similarly to the Loanable Funds theory, interest rate is determined by the equilibrium point of the market.

The diagram of Figure 4.2.2.b shows the equilibrium in the market of money. Interest rate is represented on the vertical axis, whereas money supply and demand are shown on the horizontal axis. The supply curve is vertical to the horizontal axis, and represents the stable

Figure 4.2.2.b: Liquidity Preference Theory

money supply, provided by the central bank, independently of the interest rate level, as mentioned above (this assumption consists the main difference with the loanable funds theory). Demand for money is negatively related to interest rate (because in this case interest rate is the cost of holding money against investing in a bond) and line L shows the quantity of money demanded at any given interest rate, ceteris paribus. The intersection of the two curves, M and L , represents market equilibrium and determines the level of the interest rate at r^* . When interest rate is higher than r^* , then it becomes more profitable for people to invest than to hold their money, so demand for money (cash) is lower than the quantity of money that circulates in the market (excess supply). On the other hand, when interest rate r is lower than the level of r^* , then the opportunity cost of money is lower, and people demand for more money than the central bank currently supplies (excess demand). When $r=r^*$, then people want to hold exactly the amount of money they currently have and equilibrium is achieved. In case of a change in demand for money because of a change in another determining factor, e.g. income, or in case of a change in the quantity of money supplied by the central bank, the equilibrium rate will change. For example, if the central bank decides to increase money supply, then M would increase to M_1 and the curve in the diagram of Figure 4.2.2.b would shift to the right. Consequently, equilibrium would be defined by point E1 and the new interest rate in the market would be r_1 , lower than r^* . On the one hand, individuals or enterprises, who want to invest, form the demand for loanable funds. They ask for loans in order to proceed with an investment. As interest rate gets higher, borrowing becomes more expensive. As a result, demand for loanable funds decreases as interest rate increases. On the other hand, the supply of loanable funds comes from people or enterprises that use the loanable funds market to save their money. Instead of consuming part of their income, they choose to put it into the loanable funds market in order to save it for later. In this case, higher interest rate means higher return on savings. Therefore, supply of loanable funds rises as interest rate increases.

4.3 Empirical Methodology

This work is a systematic literature review that synthesizes knowledge from existing primary studies. The systematic literature review has been designed and executed according to the guidelines presented by Kitchenham et al. (2009). The goal of this study is to develop the Framework for managing Interest in Technical Debt (FITTED). Based on the framework's goals (see Chapter 4.1), we set three research questions:

RQ1: *What are the types of TD interest and the parameters of its valuation?*

RQ2: *How characteristics of interest in economics are applied in TD interest?*

RQ3: Can we define a TD interest theory?

In order to select the studies to be explored in this research effort, we consider the primary studies investigated in Chapter 2) and by Li et al. (2015) (see Chapter 4.2.1). Both these secondary studies are recent and have explicitly investigated the concept of interest, so they can provide a valid and up-to-date set of primary studies. We considered the union of the two sets of primary studies and select those that are related to interest. On the completion of this process, we ended up with 36 studies that focus on TD interest. In Table 4.3, we present the reference numbers of the selected studies (as provided by the original secondary studies). In the last column, the total number of primary studies retrieved from every literature review is provided. In the parenthesis, we note the number of studies reported by only one of the two literature reviews, also indicated by the boldface reference numbers. Note that the two secondary studies have obtained different sets of primary studies since they applied different inclusion criteria (e.g., Ampatzoglou et al. (2015a) excluded all studies that were not using any material from finance).

Table 4.3: Studies Included in this Review

Source of Reuse	Reference in the original secondary study	Study Count
Ampatzoglou et al. (2015a)	P1, P2, P7, P8, P9, P10, P12, P13, P15, P16, P17, P22, P23, P24, P26, P30, P35, P37, P38, P41, P42, P51, P52, P53, P57, P58 , P67, P68, P69	29 (1)
Li et al. (2015)	S1, S2, S9, S10, S11, S12, S15 , S16, S17, S20, S22, S23, S24, S34, S35, S39, S40 , S42 , S50, S53 , S55, S56, S61, S62, S65, S71, S72, S73, S75 , S77 , S78, S85 , S91, S93, S94	35 (7)

The study selection process of the two secondary studies surveying primary studies on TD can be summarized as follows:

Ampatzoglou et al.

- Queried 7 digital libraries (IEEE, ACM, Scopus, Springer, Science Direct, Web of Science, and Google Scholar), with the term technical debt. The search returned 1,173 primary studies
- Applied Inclusion/Exclusion Criteria (e.g., the study should focus on the financial aspect of TD). The process returned 69 primary studies.

Li et al.

- Queried 6 digital libraries (IEEEExplore, ACM, Science Direct, Web of Science, Springer Link, Scopus, Inspec, with the term debt, in the area of computer science or software engineering. The search returned 1,665 studies

- Applied Inclusion/Exclusion Criteria (e.g., paper should focus on some specific types of TD). The process returned 94 primary studies.

To answer the RQs and construct FITTED (see Chapter 4.4), we extracted data from the primary studies and combined this information with theory from Economics (see Chapter 4.2.2). In particular, in order to answer **RQ1**, we have isolated all definitions of interest, and the parameters used for its valuation. To answer **RQ2**, we have extracted and analyzed all statements regarding how interest evolves over time. To answer **RQ3**, we have looked into the accumulation and repayment of interest, as well as the Liquidity Preference Theory. Eventually we performed a synthesis of the data in the primary studies and the aforementioned economics theory. Therefore, to some extent, FITTED represents our own ideas that aim at covering gaps in literature.

4.4 Framework for Managing Interest in TD

In this chapter, we present FITTED, i.e., the proposed framework for managing interest in technical debt, structured based on the set RQs.

4.4.1 Types of Interest and Parameters of its Valuation

According to Ampatzoglou et al. (2015a) and Li et al. (2015), interest is the most prominent financial term that is used in TDM research. Note that in economics, interest theories are used for calculating interest rate (not interest per se), since interest is calculated based on interest rate. However, in TDM, interest is not calculated based on interest rate, but it is assessed in various other ways, as explained later in this chapter. Specifically, from TD research, it is not clear if interest rate can be defined at all. Based on the selected primary studies, technical debt interest is perceived as a risk for software development, in the sense that it has a specific effect (i.e., *interest amount*) and a probability to occur (i.e., *interest probability*). Concerning these two terms:

- 28% of primary studies describe *interest amount* as the extra effort during maintenance, whereas 50% as the extra maintenance cost. However, since in software economics, cost is usually defined as a function of effort, we can safely assume that 78% of studies refer to interest amount as the extra effort/cost that is required during maintenance activities, due to the presence of technical debt. The rest of the studies, either provide more high-level definitions—e.g., (Eisenberg, 2012) and (Letouzey, 2012)—or define technical debt interest similarly to economics, i.e., the increase rate of technical debt amount (Ernst, 2012), or define interest as a change in a design-time quality attribute—see for example (Seaman et al., 2012) and (Zazworka et al., 2011).
- 25% of the studies acknowledge the existence of *interest probability*. According to

Seaman and Guo (2011), interest probability refers to the possibility that interest will occur in one item, i.e., that extra effort will be demanded during future maintenance. It should be noted that Technical Debt Items might not be need to be maintained and thus no interest will be paid. From the studies that deal with interest probability, (Guo and Seaman, 2011) and (Snipes et al., 2012) adopt a financial risk management approach where interest probability is calculated as the standard deviation of past interest rate; (Izurieta et al., 2013) suggests that interest probability is time sensitive; the rest, as (Guo et al., 2011) or (Seaman and Guo, 2011), adopt a risk management approach, i.e., they consider interest probability as the probability of the TD incurring event to occur.

In addition to the amount and probability of incurred interest, it is useful to distinguish between two different situations when interest can manifest itself:

- ***Interest while performing maintenance activities—IM:*** Performing maintenance tasks is more time/effort consuming in software with accumulated TD, compared to the same software if it had no TD (see for example (Alman, 2012), (Nugroho et al., 2011) or (Seaman et al., 2012)). The difference between the two amounts of effort is the amount of the *IM* interest. This type of interest will occur, and will be simultaneously paid, when maintenance tasks are performed (i.e., while undertaking the effort to perform the maintenance task).
- ***Interest while repaying TD—IR:*** The effort for repaying technical debt at any time point t (i.e., enhancing the quality of a Technical Debt Item – TDI to partially or totally remove TD) is higher than the effort needed for repaying technical debt for this item, at any time point prior to t , supposing that the TDI has been extended along software evolution—see for example (McGregor et al., 2012) and (Nord et al., 2012). Therefore, *IR* is calculated as the difference between the two aforementioned efforts. This type of interest will occur when (and if) the amount of TD is to be paid off. For example, consider a long method bad smell that initially consisted of 500 lines of code. During evolution, the same method grows to 750 lines. Thus, the time needed for refactoring it (TDI repayment) increases over time.

Both of the aforementioned types of interest are in agreement with the most established definitions of interest amount (i.e., extra cost/effort); however, they add more details on when these extra costs/efforts can occur. Thus, for each technical debt item, interest (I_{TDI}) should be calculated, based on the following high-level formula:

$$\begin{aligned}
 I_{TDI} &= IR_{TDI} + IM_{TDI} = \\
 &= PR_{TDI} * ER_{TDI} + PM_{TDI} * EM_{TDI},
 \end{aligned}$$

in which P denotes the probability of a repayment (R) or maintenance (M) event to occur, and E the effort needed to perform the action. For example, ER_{TDI} corresponds to the effort required for repayment in the technical debt item: TDI. To transform the aforementioned formula from the TDI level to the system-level, we propose the use of the sum aggregation function, since the total TD of a system is the sum of TD of all items with incurred TD. Therefore, interest at system level (I) can be calculated, as:

$$I = \sum_{j=1}^{j=count(TDI)} PR_j * ER_j + PM_j * EM_j$$

We note that to use the aforementioned formulas, one needs to assign estimates for the PR , PM , ER , and EM factors; these can either come from experience or from empirical data. For examples and interesting research directions on this issue, see Chapter 4.5.

4.4.2 Mapping of Interest Characteristics in Economics on TD Interest

Based on economics, interest is classified over two dimensions: its method of calculation and its variation over time. For these purposes, interest or interest rate can be:

- **Simple or Compound:** Interest is simple when it is calculated only as a function of the principal; whereas it is compound when it is calculated over the principal, plus the incurred interest; and
- **Fixed or Floating:** Interest rate is fixed, if it does not change along time; whereas it is floating when it can increase or decrease based on circumstances.

The primary studies we looked into, discuss these characteristics of interest, but only superficially, without empirical evidence on the real-world evolution of interest. As already explained in Chapter 4.4.1, the concept of interest rate does not apply for technical debt. Therefore, the distinction between floating and fixed interest rates is not applicable. Specifically, 19% of primary studies deal with the evolution of interest along time and either characterize it as compound, or continuously increasing. As an exception to this, Chin et al. (2010), propose that the cost of the organization to hold on TD is stable across time and neither increases nor decreases. Finally, only 14% of the studies propose a specific way of measuring interest. The estimation is commonly performed by using historical data, documentation, and maintenance models.

From the primary studies, we observe that technical debt interest is perceived as *compound*, in the sense that it is increasing, since the additional effort to repay technical debt and perform maintenance on a technical debt item increases as software grows and design-time quality is compromised (see for example (Allman, 2012) and (Buschmann., 2011)). It is suggested that the level of compromised design-time quality (CQ) (e.g. higher complexity

or coupling) can be used to calculate the increase of interest. However, it is rather problematic to decompose the level of compromised design-time quality of the system to the level of the original system at time t_1 , $CQ(t_1)$, i.e., the one that existed in the system when the principal incurred, and the additional level of the system at (t_2) , i.e., the one that incurred due to system evolution (system larger in size, more functionality, etc.): $\Delta Q = CQ(t_2) - CQ(t_1)$. Therefore, the calculation of the effort needed to perform any maintenance action in t_2 , can only be assessed based on current level of compromised design-time quality of the system $CQ(t_2)$. However, in case that some repayment activity is performed (at t_2), we expect that the level of compromised design-time quality of system after partial repayment $CQ(t_2)$ to decrease (i.e., $CQ(t_2) < CQ(t_1)$), leading to a decreased amount of both types of interest, in future maintenance—i.e., ER and EM . These claims are valid for TDIs, in which no additional technical debt has been incurred between timestamps t_2 and t_1 .

4.4.3 Breaking Point Theory

Based on the above, and by borrowing the rationale of the equilibrium achievement from the existing economic interest theories, we developed an interest theory for managing TD interest. Specifically, we adopt the concept of the *Liquidity Preference Theory*. The reason for selecting the *Liquidity Preference Theory* and not the *Loanable Funds Theory* is that in TD the amount of money that is available to the company for TDM is stable, i.e., the principal (the amount that has been saved, while incurring TD supposing that principal is not invested to provide extra benefits). The assumption that the available money for TDM is the principal is based on the fact that principal is the maximum amount that can be saved without spending any additional effort.

In the proposed interest theory, we map *money supply* to *principal*, in the sense that principal is the amount of money that is available to the software development company, after incurring TD; and we map the *money demand* to the *accumulated amount of interest*, in the sense that this is the extra amount of money that is demanded by the company when performing future maintenance activities, caused by the TD. In Figure 4.4.3, where we present the FITTED Interest Theory, the *x-axis* represents *time*, whereas the *y-axis* represents *amount of money*. Therefore, the equilibrium point (E_0) denotes the time stamp (t_0), in which the company has spent the complete amount of money from the internal loan (i.e., initial principal $- P_0$) in extra maintenance activities because of the incurred TD. We note that the specification of the equilibrium point is achieved through an analysis based only on effort, i.e., the effort saved when taking on TD and the extra effort required for any future maintenance activity because of its accumulation. Any other related costs or benefits related to technical debt occurrence (e.g. gains from launching the product earlier) have been excluded from the model for simplicity reasons. Thus, if the expected lifespan of the spe-

cific TDI is shorter than t_0 then undertaking technical debt is a beneficial choice, whereas if not, technical debt becomes harmful for the company. The aforementioned discussions, in the case that no repayment actions are performed, are summarized in the blue lines of Figure 4.4.3.

Figure 4.4.3: Breaking Point Theory

Additionally, in Figure 4.4.3, we consider $\Sigma(IM)$ as *continuously increasing*, since it is a sum of positive numbers and because TD interest is compound (see Chapter 4.4.2). In case that some repayment occurs at some timestamp (t_r), the line of the accumulated interest $\Sigma(IM)$ is moved upwards, due to the interest paid for repayment – i.e., $IR(t_r)$ —but its slope is decreasing, since the interest is expected to lower for future maintenance activities (IM). This in turn leads to a shift of the equilibrium point (E') to the right, increasing the benefit period (t_0'). The money supply line (P_0) is not moved, because the originally available budget of the company is not affected, assuming zero investment or inflation. The proposed theory can help practitioners in their decision making by:

- Identifying the timestamp in which incurring TD, becomes harmful for the company. Thus, they can decide if they should undertake the debt.
- Supporting the continuous monitoring of the interest that has been paid so far.
- Evaluating the repayment activity, based on the time-shift of the equilibrium point that it offers.

4.5 Research Implications

Research on TD is theoretical and lacks empirical evidence. According to Li et al. 49% of the complete corpus of TD research presents no empirical evidence, or only toy examples,

whereas this number rises to 56%, when focusing on interest (Li et al., 2015). Thus, in this chapter we provide future work directions, which could boost empiricism on TD interest and support the realization of FITTED.

Types of Interest and Evolution:

- Which one of the two types of interest (*IR* or *IM*) produces higher interest amounts when it occurs? Answering this question would suggest which type of interest would be more profitable to manage.
- How can *IR* and *IM* amount be modelled, as a function of the principal, or the underlying structure of the TDI? Answering this question would open new research directions in TD management, since more elaborate management approaches could be employed.
- What is the relationship of the decay of quality in the underlying system and the increase in *EM* or *ER*? Answering this question could guide practitioners on how to model the increase of interest during software evolution.

So far, these questions have been partially explored by Guo et al. (2011), Nugroho et al. (2011), and Siebra et al. (2012), by exploring historical changes and documentation. The research state-of-the-art lacks real-world evidence on effort allocation.

FITTED Interest Theory:

- What is the expected time (t_0) when the equilibrium will be reached?
- What is the average time-shift from performing specific repayment activities?
- What factors influence this time-shift?
- What is the relationship between *IR* and the average decrease in the *IM* of future maintenance activities?
- Answering these questions, would enable researchers to instantiate the proposed theory, based on context-specific data, and offer FITTED as a useful tool for practitioners (see Chapter 4.6).

4.6 Illustrative Example

As an illustrative example of how the FITTED interest theory can be applied, Chatzigeorgiou et al. (2015) proposed an *empirical setting for calculating the expected time (t_0) when the equilibrium will be reached* (see first question on FITTED Interest Theory in Chapter 4.5)⁸.

Based on this study, the time point at which the equilibrium is reached has been termed as the *breaking point of technical debt*.

To instantiate this approach with a specific implementation, Chatzigeorgiou et al. (2015)

⁸ We note that for this illustration we have not considered the equilibrium point shift that is described in FITTED, but we only calculate the equilibrium point without any refactoring and we assume that while adding features quality does not further decay.

performed the following actions/assumptions:

- **Fitness function (FF):** Ratio between coupling and cohesion, i.e., two well-known software quality characteristics.
- **Principal:** Number of simple refactorings (move methods, extract class, etc.) needed to transform the actual design to the optimal one. The cost of a simple refactoring according to the literature is 5.73\$ (Wake, 2003).
- **Interest:** The ratio of the levels of design quality (fitness function) is correlated to the ratio of the two EM (the actual and the optimum). Thus, and by considering that on average the addition of one line of code costs 1.83\$, interest can be calculated as $1.83 * LOC_{added} * (1 - FF_{ratio})$.

The application of this approach in an open-source system, namely Junit, has shown that an interest of 243\$ is accumulated per release and its principal in version 4.10 (starting version) is 1,891\$. Therefore, the breaking point will be reached in approximately 8 versions after the start version, i.e., about 5.5 years from the time of analysis (see Figure 4.6.b).

We note that for simplicity, Chatzigeorgiou et al., have assumed that interest is a linear function of the FF, therefore, the growth of interest, i.e., $\Sigma(IM)$, is linear as well, in contrast to Figure 4.4.3.

Figure 4.6.b: Estimating the Breaking Point of JUnit

Consequently, if no TD repayment activities are performed in this time period the difficulty of maintaining this system (interest) will be as high as the effort saved from not having developed the optimal system (principal). However, we need to note that 5.5 years is a long period, and thus, the project is considered to have a sustainable amount of TD.

4.7 Threats to Validity

In this chapter, we report threats to the validity of this research. Since some parts of the research method application are inherited from the secondary studies of Ampatzoglou et al. (2015a) and Li et al. (2015), their threats to validity are also inherited. Therefore, the following threats have been identified:

Possibility of missing primary studies. We believe that although this threat still exists, we have substantially mitigated it, by considering the union of primary studies reported on the two secondary studies. Also, the fact that their overlap was quite high indicates that their coverage was adequate.

Publication bias. As both secondary studies report, in many cases research areas in their infancy are biased by the fact that most of the studies are published by a small community. However, we believe that this fact is currently changing in TD, since more and more researchers are getting actively involved in the last years.

Concerning data extraction, since we independently performed this step, the corresponding threats are related only to this study. To mitigate bias while extracting data, two researchers performed data collection independently, compared the results and discussed possible differences. The final dataset was built through the consent of all authors. Finally, as a threat we acknowledge that the construction of the presented formulas, is to some extent based on the understanding of the authors on TD interest.

4.8 Conclusion

Nowadays, Technical Debt (TD) is receiving increasing interest by both academia and practitioners, leading to an explosion of studies in this field. The cornerstones of TD are two notions borrowed from economics: i.e., principal and interest. Although principal is a well-established term, interest has so far been discussed in a rather coarse-grained way, with several contradictions among researchers. In this study, we propose FITTED, i.e., a framework for managing interest in TD, which takes into account existing TD literature and economic interest theories. The framework comprises of: (a) the parameters that are used in TD interest valuation, (b) a classification of TD interest types, (c) a characterization of TD interest evolution, and (d) a TD interest theory, based on the Liquidity Preference Theory. The proposed framework is expected to aid in the decision making of practitioners, and points to interesting research directions.

Chapter 5 – A Framework for Managing Interest in Technical Debt: An Industrial Validation

In the previous chapter we introduced a framework for calculating the Technical Debt Breaking Point (TD-BP), which is a point in time where the accumulated interest becomes larger than the principal; thus, the debt of the company is no longer sustainable after this point in time. In this chapter, we instantiate this framework and validate its ability to assess the breaking point of source code modules in an industrial setting. The results of the validation suggest that the calculated TD-BP is strongly correlated to experts’ opinion on the sustainability of modules, and that it can accurately rank components, based on their maintenance difficulty.

5.1 Motivation

In the last years, the technical debt research community has been spending increasingly large effort on the different activities of Technical Debt Management. To a large extent, this research is focusing on issues related to cost and benefit analysis of software quality management (Lim et al., 2012). An overview of TDM research is presented in, among others, two literature reviews (Ampatzoglou et al., 2015a), (Li et al., 2015). Although research on TDM has been intense in the last years, the community still faces major challenges, two of which are the focus of this study: (a) a sound estimation of the amount of TD (i.e., the quantification of interest and principal (Ampatzoglou et al., 2015a)), and (b) the monitoring of the TD amount, as it increases due to the accumulation of interest (Ampatzoglou et al., 2015b).

In the previous chapter, we begun to address these challenges, by defining a theoretical framework, named FITTED, that can be used for the long-term management of Technical Debt. FITTED considers that interest accumulating during software evolution, can potentially outgrow the amount required for repaying the principal of TD. Therefore, it is critical for project managers to be able to estimate the point in time where the accumulated interest will be equal to the TD principal. This point in time is called the “breaking point” (TD-BP), in the sense that any benefit deriving from the decision to take on technical debt is being neutralized after that point, i.e. the cost becomes higher than the benefit (Ampatzoglou et al., 2016b).

The aim of this study is twofold: (a) we instantiate the FITTED framework by providing details on the calculation of principal and interest, and (b) we validate the proposed instantiation of the FITTED framework in an industrial setting. We note that both the instantiation of the framework and its subsequent validation are at the source code level; instantiation could also take place for technical debt at other levels but that is out of the scope of this study. The instantiation of the framework is supported by a software tool which can extract data on: (a) the principal of technical debt, (b) the interest accumulated in every version of the system, and (c) the occurrence of the breaking point for every version of the system.

The validation of the framework was conducted in collaboration with a software development company. Specifically, the tool was applied on two of their current software development projects, and its results were analyzed together with two software engineers. Compared to the state-of-research (see Chapter 5.2.1), this is the first automated approach that calculates the Breaking Point of Technical Debt and validates it within an industrial setting. The rest of this chapter is structured as follows: In Chapter 5.2, the FITTED framework is presented. In Chapter 5.3, we present our approach for validating FITTED, and in Chapter 5.4, we explain the design of our study. Next, we present the results of our analysis, in Chapter 5.5, and we discuss them, in Chapter 5.6. Finally, in Chapter 5.7 we refer to the threats to validity.

5.2 FITTED Interest Theory

To develop a theory on managing technical debt interest, we borrowed the rationale of the economic interest theories on the equilibrium achievement. More particularly, we adopt the idea of the breaking point at the money market, where money supply is equal to money demand. As long as accumulated interest is lower than the principal, the benefit derived by the initial decision to save effort is more than the cost generated by the extra effort to maintain the software. On the other hand, when the accumulation of interest overcomes the principal, then the cost needed for the system's maintenance becomes more than the money saved when technical debt incurred. Consequently, the point where the accumulated interest is equal to the principal—breaking point (Chatzigeorgiou, 2015) is very critical in technical debt management and can help project managers in their decision-making. Under this perspective, we map principal to money supply – since it represents the amount of money available – and the accumulated amount of interest with money demand – since it embodies the extra amount of money that will be necessary for future maintenance activities caused by the TD.

Figure 5.2.2 represents the FITTED Interest Theory: the horizontal axis depicts time, while the vertical axis stands for money amount. The blue curve $\Sigma(I)$ denotes the increasing cumulative interest, as the software develops. We consider cumulative interest as continuously increasing, because it consists of the accumulation of interest in every successive version of the project and because TD interest is compound (Ampatzoglou et al., 2015b). Principal, on the other hand, is represented by the green line P and is also considered as increasing, in the sense that the introduction of new functionality to the software generates new TD items. Concerning principal, we have to point out that initially (see Chapter 4.4.3), we defined it as an amount of effort saved once while taking on technical debt and considered it stable throughout software evolution. However, as software grows, technical debt keeps accumulating. Therefore, in this study we take into account the development of principal and we calculate it during software development, so that we are able to reach a more accurate assessment of the version where breaking point occurs.

Figure 5.2.2: Updated FITTED Interest Theory

As described by the figure, until time stamp t_0 , interest is lower than the principal and taking on technical debt can be perceived as beneficial for the project. After time point t_0 , interest accumulates and becomes larger than the principal and extra maintenance costs exceed the initial money saving. Therefore, the equilibrium point E , denotes the point in time t_0 where the complete amount of money saved by incurring TD (i.e., the principal) equals the money spent on extra maintenance activities caused by the incurred TD (Ampatzoglou et al., 2016b). If the development team proceeds with some repayment activity, e.g. at timestamp t_r , accumulated interest increases, because of the addition of the repayment effort, and line $\Sigma(I)$ shifts upwards to $\Sigma(I)'$. However, the slope of the line decreases, since lower maintenance activities are expected in the future due to the repayment. On the other

hand, principal decreases at time stamp t_r , because of the repayment activity, and then follows the course of line P' . Consequently, the equilibrium point moves to the right E' , and the benefit period increases to time stamp t_0' (Ampatzoglou et al., 2016b).

5.3 FITTED Instantiation

In order to instantiate the aforementioned approach, we need to describe the way that the two components of TD-BP can be calculated. To this end, in Chapter 5.3.1 we describe the employed way for calculating principal at source code level, whereas in Chapter 5.3.2, we present the proposed approach for assessing interest on source code artifacts.

5.3.1 Principal Estimation

As proposed in literature, TD principal can be calculated as a function of three variables (Curtis et al., 2012). The first variable is the number of problems that must be fixed, the second one is the time required to fix each one of these problems, and the third one is the cost for fixing each problem. Regarding the number of must-fix problems, in our previous work (Chatzigeorgiou et al., 2015) we have presumed that there is an actual design quality for any object-oriented software, which can be estimated by a proper fitness function. Under this perspective, the ‘spread’ between the optimal and the actual design, as it can be derived by the difference in their fitness function values, can be mapped to the principal, i.e., the effort needed to convert the actual system to the correspondent optimum one. This notion of ‘distance’ is depicted in Figure 1a (see Chapter 1.1), as Effort.

For the purpose of this study, we have decided to estimate principal, based on the computation of a widely used OSS platform, SonarQube—formerly known as Sonar, which is a state-of-the-art tool for calculating TD principal. According to its documentation (SonarQube, 2018), SonarQube aims at the continuous evaluation of software quality. SonarQube can assess the quality of software on a plentitude of programming languages, generating documentation on quality measures and issues, such as coding rule violations. The analysis has been performed according to the platform’s default format, without any further configuration. The platform algorithm is based upon an adopted version of the SQALE method proposed by Letouzey and Coq (2010), in which a remediation index is obtained for requirements of an applicable Quality Model. For example, for a requirement stating that all files should have at least 70% code coverage, the corresponding remediation action is to write additional tests. A remediation function maps effort to each action, for example, 20 minutes per uncovered line of code. Finally, for each artifact, the remediation index relating to all the characteristics of the Quality Model is obtained by adding all remediation indices linked to all quality requirements. The resulting SQALE Quality Index is considered to represent the principal of the TD for the assessed source code.

5.3.2 Interest Estimation

In this chapter, we discuss the way we calculate interest in the proposed approach. In particular, in Chapter 5.3.2.1 we present the theoretical background of the approach, whereas in Chapter 5.3.2.2, we present the metrics and the method that we have used for its instantiation at the source code level, also known as code TD.

5.3.2.1 Mathematical Background

Assuming the optimal and actual systems of Figure 1a (see Chapter 1.1), in the common case, maintaining the optimum system requires less effort than maintaining the actual system. As shown in the figure, adding a new feature A to the optimum system needs a certain effort, noted as $Effort_{(optimum)}$, whereas adding the same feature to the actual system necessitates a larger effort, noted as $Effort_{(actual)}$. The difference between these two efforts represents the interest that is accumulated during this maintenance activity, i.e., the addition of feature A.

However, since the evolution of the software cannot be predicted, it is not possible to foresee what kind of modifications will be made in a system during future releases. Hence, we base our assessment of future maintenance effort on historic data, by considering past effort spent on maintenance activities. More specifically, although many measures can be used for the calculation of past effort, we have selected to use the average number of lines of code added between sequential releases. Added lines of code indicate the amount of effort required for the addition of new functionality and in a way the effort needed for changing existing modules. Supposing that an average of k lines of code is added in the transition from one version to another, then we can accept that adding k lines to a high design quality system (as denoted by the assigned fitness function value) is easier than making the same addition to a lower design quality system. At this point we suppose that maintenance effort is dependent on the design quality, as in (4):

$$Effort_m = c \cdot FitnessVale \quad (4)$$

where c is an arbitrary constant. Based on the abovementioned approach, the fitness value for both the optimum and the actual systems can be calculated and therefore we can capture the proportion of the theoretical over the actual effort, as in (5):

$$\begin{aligned} \frac{Effort_m(optimum)}{Effort_m(actual)} &= \frac{c \cdot FitnessVale(optimum)}{c \cdot FitnessVale(actual)} \Rightarrow \\ Effort_m(optimum) &= \frac{FitnessVale(optimum)}{FitnessVale(actual)} \cdot Effort_m(actual) \end{aligned} \quad (5)$$

Hence, if past maintenance effort (i.e., the number (k) of added lines) is used to define the actual maintenance effort, then the optimum effort can be directly obtained. Consequently,

the amount of interest accumulated between any two sequential versions can be calculated as the distance between the optimum and the actual effort and is given by (6) and (7).

$$Interest = \Delta Effort = k \cdot \left(1 - \frac{FitnessValue(optimum)}{FitnessValue(actual)} \right) \quad (6)$$

$$Interest = \Delta Effort = k \cdot \left(\frac{FitnessValue(optimum)}{FitnessValue(actual)} - 1 \right) \quad (7)$$

Since the optimum value for each one of the applied metrics could be either the minimum or the maximum value, interest is given by the actual effort (k) multiplied by 1 minus the ratio of the optimum fitness value to the actual fitness value, if the optimum value is the maximum one—as in (6). Otherwise, if the optimum value is the minimum one, interest is given by the actual effort (k) multiplied by 1 minus the ratio of the actual fitness value to the optimum fitness value—as in (7).

5.3.2.2 Interest Calculation for Source Code Debt

Given the fact that interest is closely related to the system’s maintainability (Ampatzoglou et al., 2016a), our analysis concentrates on the estimation of interest, based on well-known maintainability models. Maintainability models are sets of low-level metrics that quantify design-time quality attributes, such as inheritance, cohesion, coupling, complexity and size, which are somehow aggregated in one maintainability index. Therefore, given the relation between maintainability and interest, in this study we define the fitness value used for the estimation of the interest, using the set of metrics that belong to well-known metric suites (Chidamber et al., 1998), (Li and Henry, 1993) and are accurate maintainability predictors. Riaz et al. (2009), in their systematic review have studied a variety of existing maintainability models, and reported on their accuracy. Based on the results of that systematic literature review, a list of ten metrics stands out as successful maintainability predictors (see Table 5.3.2.2)—and this is the set of metrics that we use in our study. Table 5.3.2.2 depicts these metrics and the quality attribute to which they refer, as presented by Arvanitou et al. (2016). Nevertheless, we need to note that in our case no aggregation function is used for these metrics, since it is out of the scope of our analysis.

Table 5.3.2.2: Object-Oriented Metrics

Metric	Description	Quality Attribute
DIT	<i>Depth of Inheritance Tree</i> : Inheritance level number, 0 for the root class.	<i>inheritance</i>
NOCC	<i>Number of Children Classes</i> : Number of direct sub-classes that the class has.	<i>inheritance</i>
MPC	<i>Message Passing Coupling</i> : Number of send statements defined in the class.	<i>coupling</i>

Metric	Description	Quality Attribute
RFC	<i>Response For a Class</i> : Number of local methods plus the number of methods called by class methods.	<i>coupling</i>
LCOM1	<i>Lack of Cohesion of Methods</i> : Number of disjoint sets of methods (a set of methods that do not interact with each other), in the class.	<i>cohesion</i>
DAC	<i>Data Abstraction Coupling</i> : Number of abstract types defined in the class.	<i>coupling</i>
WMPC	<i>Weighted Method per Class</i> : Cyclomatic complexity of all methods in the class.	<i>complexity</i>
NOM	<i>Number of Methods</i> : Number of methods in the class.	<i>size</i>
SIZE1	<i>Lines of Code</i> : Number of semicolons in the class.	<i>size</i>
SIZE2	<i>Number of Properties</i> : Number of attributes and methods in the class	<i>size</i>

More specifically, in order to assess the optimum design, our analysis consists of the following steps. Firstly, classes of high similarity level with the investigated class are found (see Chapter 5.3.4). The identification and comparison of a class with structurally similar ones or the calculation of the fitness function is crucial, in the sense that the maintainability of a class of tens of LoC cannot be comparable with one with hundreds of LoCs. Secondly, the optimum value for each one of the aforementioned metrics is identified, the difference between the actual and the optimum fitness values is calculated and the average distance between the actual and optimum design is assessed. Subsequently, the interest is calculated as described above, in equations (6) and (7). Finally, taking into account the number of LoC that can be added in a unit of time (according to past data) and the developer's hourly rate, interest is estimated in terms of money.

5.3.3 Breaking Point

According to the abovementioned analysis, an estimate of the interest in USD can be achieved, based on the difference between theoretical effort (maintenance performed on the optimum system) and the actual effort, on past data concerning the time required for the addition of lines of code, and the per hour cost of a developer's work. Therefore, given the principal, as it comes from SonarQube, and knowing the interest accumulated in one version of the software, the ratio of the principal over the interest gives the number of versions after which the company will reach the breaking point, as shown in (8).

$$TD_{BP} = \frac{Principal(\$)}{Interest(\$)} \quad (8)$$

5.3.4 Tool Support

To boost the applicability of the aforementioned methodology, and moreover to validate the FITTED framework, we created a desktop application, developed in Java, called Breaking Point Calculator (BPC). The procedure is separated into two phases, the analysis phase and the computation phase. During the analysis phase, the source code of the project is analyzed and the metrics needed for the calculation of the results are retrieved. Each project is analyzed with the aid of two software tools, SonarQube and Percerons Client:

- **SonarQube** is used for the extraction of five metrics which are used as a factor of similarity and technical debt principal. The metrics used for identifying similar classes are depicted in Table 5.3.4, accompanied by their description. The analysis of a project by SonarQube can take several hours, thus it takes place on our server in UoM (University of Macedonia) where it is installed. After the analysis is complete the metrics are stored in SonarQube database, which we access to retrieve them.
- **Percerons Client** is a toolset⁹ developed to support empirical software engineering research. Percerons is able to calculate metrics for every class contained in a .JAR file and produces a .CSV output (Ampatzoglou et al., 2013). Percerons Client is used to extract the object-oriented metrics of Table 5.3.2.2 for each class, on every version of the investigated project. The output document is parsed by BPC and the metrics of each class are stored into a database.

Table 5.3.4: Similarity Metrics

Metric	Description
Classes	Number of classes (including nested classes, interfaces, enums and annotations).
Complexity	The average complexity calculated based on the number of paths through the code.
Functions	Number of functions.
LOC	Number of physical lines that contain at least one character which is neither a whitespace nor a tabulation nor part of a comment.
Sqale Index	Effort needed to fix all maintainability issues.
Statements	Number of statements.

Once the analysis phase is completed, we proceed with the computation of the breaking point, by selecting the project under investigation, along with the computation scope, i.e., whether we want to perform the computation on “class” or “package” level. Classes and packages selected for our scope are being referred to as artifacts. The goal of the computa-

⁹ <https://extreme.se.uom.gr/>

tion phase is the calculation of the breaking point. To this end, we retrieve all the artifacts for each version of the project selected, along with the similarity and object-oriented metrics that characterize each one. For every artifact under study, we detect the most similar ones by using the aforementioned similarity metrics. Thereafter, we calculate the optimal artifact by selecting the best values among the grouped artifacts.

On the completion of this phase, we are able to proceed with the calculation of the fitness value, the principal, the interest and the breaking point. To sum up, by the time the computation phase is completed, the following information can be extracted for any artifact from any available project version:

1. The five most similar artifacts to the investigated one, along with the values of the similarity metrics and the similarity rate.
2. The values of the object-oriented metrics for the most similar artifacts and the investigated one.
3. The values of the quality metrics for the optimum artifact.
4. The principal in USD.
5. The interest in USD.
6. The breaking point in versions.

Finally, we are producing overview reports for the project in .CSV documents in order to get a more comprehensive picture of it, as well as to enable the comparison of multiple artifacts. Some screenshots of the tool are provided in Figure 5.3.4.

Figure 5.3.4: Breaking Point Calculator

5.4 Empirical Methodology

To validate the proposed framework and its ability to estimate the TD-BP, we performed an industrial case study in a small-medium enterprise (SME) in Greece, which is active in the domain of mobile applications. The company wants to keep its anonymity, thus all records in the dataset have been anonymized, and no personalized information can be provided, either about the company and its projects, or the case study participants. The case study is designed according to the guidelines by Runeson et al. (2012). In the next chapters, we present the four parts of our research design, i.e., objective and research question (see Chapter 5.4.1), case selection and units of analysis (see Chapter 5.4.2), data collection (see Chapter 5.4.3), and analysis (see Chapter 5.4.4).

5.4.1 Objective and Research Question

The goal of the case study is to validate the FITTED approach, regarding its ability to assess the breaking point in an industrial environment. Therefore, our study focuses in the estimation of the accuracy of the framework in assessing the breaking point. To this end, we derived the following research question:

RQ: *Is the calculated breaking point metric correlated with the effort required for maintenance activities?*

This Research Question aims at validating the proposed instantiation method of FITTED, and in particular it assesses if the TD-BP is correlated with maintenance efforts.

5.4.2 Case Selection and Units of Analysis

To collect data for our case study, we executed the *Breaking Point Calculator* Tool on the source code of two projects of the collaborating SME: (a) a lottery engine, and (b) an online betting system. Since both systems are medium scale and a manual inspection of the code might be required from the participants, we preferred to scope our analysis at high-level packages. An additional reason for this decision is that by scaling up the level of granularity for our reporting, we come closer to the architectural debt notation that has higher impact than code TD, since larger amount of TD and interest are expected to occur. Therefore, the units of our analysis are 18 packages, contained in these two different android application projects. The projects have been developed in different context and under dissimilar circumstances; hence they are expected to have different levels of accumulated TD and interest. Moreover, prerequisites for the selection of the projects were that the projects contained adequate number of versions and that the developers would be available to answer a short questionnaire concerning the development of the projects.

5.4.3 Data Collection

To answer the aforementioned research question, we performed a three-phase analysis:

- **TD Estimation.** We applied the Breaking Point Tool on the source code of the two applications, as described in Chapter 5.3.4, and obtained data on 8 versions of the two projects. From this analysis, we recorded three main variables: *interest* (I), *normalized principal per LoC* (NrP), and *breaking point* (TD-BP).
- **Extraction of Control Variables.** We compiled a list of packages, in a random ordering, and asked the developers to assign a Likert-scale score (1-5) to each one of these packages, with respect to the effort required to maintain them, and the extent to which they consider them sustainable (i.e., need to completely rewrite them, due to heavy maintenance effort). The data extraction method for this phase was a questionnaire-based approach.
- **Discussion and Interpretation of Obtained Results.** During this phase we conducted a focus group with all developers (i.e., case study participants) and discussed their rationale for ranking specific packages with high sustainability scores, and others with a lower one.

5.4.4 Data Analysis

To answer the research question, we performed a Spearman Rank correlation, between the TD estimated and the control variable. The reason for performing a non-parametric test is that the dataset was not normally distributed, whereas the reason for performing rank correlation instead of actual values correlation is that we are not interested in the predictive power of the metrics (i.e., how well they can predict the values of the sustainability), but on the consistency of the proposed metric. According to the 1061-1998 IEEE Standard for Software Metrics (IEEE, 2009), **consistency** assesses whether there is consistency between the ranks of the quality characteristic and the ranks of the metric under study. Consistency determines if a metric can accurately rank artifacts in terms of quality. The criterion is quantified by the coefficient of rank correlation.

5.5 Results

In this chapter we present the results of the empirical study, through presenting some demographics and subsequently answering the research question.

5.5.1 Study Demographics

In Table 5.5.1, we present the average descriptive statistics for the TD estimation variables and the average number of lines per class per version, between versions (a proxy of maintenance effort per version).

Table 5.5.1: Dataset Descriptive Statistics

TD Estimator	Min	Max	Mean	Std. Dev
NrP	0,138	1,163	0,690	0,302
I	0,188	179,616	28,228	55,721
TD-BP	0,085	366,279	49,602	88,572
AVG(LoC)	0,250	281,889	21,296	65,785

Furthermore, in Figure 5.5.1.a, we present the distribution of participants' responses with respect to the control variable, i.e., the sustainability of each examined packages. Finally, in Figure 5.5.1.b, we present the evolution of the three TD estimators. In order to fit all estimators in the same graph (since they have different range of values), we normalize all of them against their value in the first of the examined versions (for that version all values equal 1), and then plot in a line chart their evolution.

Figure 5.5.1.a: Control Variable Demographics **Figure 5.5.1.b: TD Estimators Evolution**

Based on the results of Figure 5.5.1.a, we observe that the majority of the examined packages have been assigned an average sustainability, whereas the packages with high sustainability are more than those with low sustainability. Another observation that can be made, based on the evolution analysis of TD estimators (see Figure 5.5.1.b) is that Principal is a more stable metric (Snipes et al., 2012), i.e., provides less fluctuations, compared to interest (ranges from 100%-400% of the value of the first version) and the breaking point (ranges from 75% - 138% of the first version). Also, we can observe that the interest which increased a lot in the 2nd examined version is then decreasing along time, suggesting an improvement in the structural quality of the software. When developers were asked if they could explain the reason behind the enhancement of quality along evolution, one of them replied that: *“the quality of the code is a top priority for us. We are trying to structure our*

code in a modular manner, applying well-known patterns for that, like Model-View-Presenter (MVP¹⁰)”.

5.5.2 Answering the Research Question

As mentioned in Chapter 5.4.4, the research question is answered using correlation analysis. The results of the analysis are presented in Table 5.5.2. In the table, statistically significant correlations are denoted with grey cell shading.

Table 5.5.2: Spearman Rho: Assessment Power

Statistical Measure	NrP	I	TD-BP
Correlation Coefficient	-0,010	-0,747	,773
Sig. (2-tailed)	0,970	0,000	,000
N	18	18	18

The results of the correlation analysis suggest that both interest and normalized principal are negatively correlated with software artifacts sustainability which is considered an intuitive outcome. However, only the relation between interest and sustainability is strong and statistically significant. By assessing the correlation between TD breaking point and sustainability, we can observe that the relation is statistically significant and the strongest among the three examined TD estimators. This outcome suggests that the compound metric that we have proposed in this study is effective. This is an interesting finding in the sense that although principal in isolation is not correlated to sustainability, its inclusion in the calculation of TD-BP has increased its assessing power. The reasons that the developers used for explaining high and low values of sustainability are summarized below:

- **High Sustainability:**
 1. “Provision of limited functionality”
 2. “Only shows some data that never change”
 3. “Presents a static user interface as well”
 4. “Low coupling class which manipulates small amount of data”
 5. “Obeys to the Single responsibility”
 6. “Used to present static details about the app”
- **Low Sustainability**
 1. “High Coupling”
 2. “Handles feed from different data sources. Needs extra effort to be maintained”
 3. “Fundamental class of the app which handles many data and has many responsibilities”

¹⁰ MVP is a well-known pattern for mobile web development that is conceptually relevant to MVC. [Online reference](#).

5.6 Discussion

In this chapter, we interpret the case study results and discuss their implications for researchers and practitioners.

5.6.1 Explanation of Obtained Results

According to the results of the study interest (as calculated by the proposed approach) appears to be more sensitive than principal (as calculated by SonarQube) in the sense that it exhibits more fluctuations over the course of the project's evolution. This could be attributed to the fact that principal depends on the number of rule violations that SonarQube identifies, and these violations do not change drastically from one version to the other. On the other hand, interest as calculated by the proposed FITTED approach, takes indirectly into account many structural characteristics of software, as the 'optimum' design is assessed by identifying structurally similar classes. For example, even a change in the control flow structure of a method will be reflected on interest, whereas it could go unnoticed by the principal calculation. Thus, interest calculation is subject even to minor changes in the code leading to a more sensitive measure.

Considering the results of the correlation analysis between principal, interest, breaking point and software artifacts sustainability, it appears that the consideration of the structural characteristics on top of the TD issues identified by SonarQube has been successful. In other words, the combination of interest (which has a statistically significant correlation to sustainability) and principal for the extraction of the TD breaking point, leads to even stronger, and statistically significant correlation between the breaking point and sustainability.

It should be noted that in this study sustainability is related to the difficulty of performing maintenance or conversely the need to completely rewrite software modules. A module has low sustainability if it undergoes frequent and extensive maintenance and at the same time its TD hinders maintenance. According to the responses of the study typical symptoms of a poor design, such as high coupling and violation of the single responsibility principle contribute to low sustainability. The fact that the proposed TD breaking point captures to a sufficient extent the notion of sustainability as perceived by the developers implies that TD issues combined with structural properties of the code can serve as a valid indicator of maintainability.

5.6.2 Implications for Researchers

Considering the need to assess the sustainability of software modules in an objective and accurate manner, the promising results regarding the correlation between TD-BP and sustainability should be checked against more projects. Given that the case study has been

executed on two projects from the mobile domain, extension of the study to projects with different characteristics in terms of application domain, programming language and development process can be very valuable for tuning the calculation of the breaking point. It should be noted that the proposed application of FITTED targets code TD; however, the underlying theory can be extended to other levels such as architecture. Considering the architectural TD has a larger impact on system maintainability than code TD, obtaining the interest of an architectural element based on the notion of ‘distance’ to similar, ‘optimum’ elements can remove the existing barriers of effectively calculating a pragmatic TD measure.

5.6.2 Implications for Practitioners

The results of the study and the developers’ feedback on the causes of low and high sustainability indicate that the adoption of rather ‘classic’ well-known practices (such as the use of patterns, conformance to the Single Responsibility Principle, low coupling and high cohesion) can improve sustainability. FITTED confirms that addressing code and design inefficiencies can move the breaking point, i.e., the time at which increased maintenance costs exceed any short-term benefits, further away to the future.

The proposed toolset in this study and especially the consideration of structurally similar modules that exhibit a higher quality can be of help to novice software engineers. Assessing principal, interest and breaking point in a relative manner that is by comparing a software module to that of peers can yield feasible opportunities for improvement. Moreover, the proposed toolset offers a viable way of continuous monitoring of TD critical levels, so as to act as a Quality Gate in the context of contemporary continuous integration practices.

5.7 Threats to Validity

The results regarding the correlation between interest and breaking point and sustainability are subject to external validity threats meaning that they cannot be generalized to other application domains, programming languages or project sizes. Furthermore, TD principal estimation with the use of SonarQube has to be reported as a possible validity threat. Since the tool does not consider architecture when calculating TD principal, this may impact the precision of the estimation. Moreover, it should be noted that the developers in the study might have perceived the concept of sustainability in different ways. Since sustainability is one of the main variables in the analysis, construct validity threats arise, although an attempt to clarify the concept prior to the collection of results has been made.

5.8 Conclusion

Principal and Interest are key concepts in the Technical Debt literature as they provide a monetary representation of detectable inefficiencies. However, their value has to be demonstrated by proving their correlation to external software qualities. In this paper we demonstrate the feasibility of instantiating a theoretical framework which allows the calculation of the breaking point, which is a time stamp in which the accumulated interest becomes larger than the principal. The results of a validation in an industrial setting are very promising since they suggest that breaking point is strongly correlated to experts' opinion on the sustainability of modules.

Chapter 6 – Exploring the Relation between Technical Debt Principal and Interest: An Empirical Approach

The cornerstones of technical debt (TD) are two concepts borrowed from the field of economics: principal and interest. Although in economics the two terms are related, in TD there is no study on this direction so as to validate the strength of the metaphor. We study the relation between Principal and Interest, and subsequently dig further into the ‘ingredients’ of each concept (since they are multi-faceted). In particular, we investigate if artifacts with similar levels of TD Principal exhibit a similar amount of TD Interest, and vice-versa. To achieve this goal, we performed an empirical study, analyzing the dataset using the Mantel test. Through the Mantel test, we examined the relation between TD Principal and Interest, and identified aspects that are able to denote proximity of artifacts, with respect to TD. Next, through Linear Mixed Effects (LME) modelling we studied the generalizability of the results. The results of the study suggest that TD Principal and Interest are related, in the sense that classes with similar levels of TD Principal tend to have similar levels of Interest. Additionally, we have reached the conclusion that aggregated measures of TD Principal or Interest are more capable of identifying proximate artifacts, compared to isolated metrics. Finally, we have provided empirical evidence on the fact that improving certain quality properties (e.g., size and coupling) should be prioritized while ranking refactoring opportunities in the sense that high values of these properties are in most of the cases related to artifacts with higher levels of TD Principal. The findings shed light on the relations between the two concepts, and can be useful for both researchers and practitioners: the former can get a deeper understanding of the concepts, whereas the latter can use our findings to guide their TD management processes such as prioritization and repayment.

6.1 Motivation

The cornerstones of the TD metaphor are two terms borrowed from the concept of debt in finance: principal and interest, which have already been defined in Chapter 5. The assessment of principal and interest depends on the type of TD (e.g., code, design, testing TD). The scope of this work is limited to *TD on the source code*, which is the most studied type of TD in the literature (Alves et al., 2016), the most supported by tools (Alves et al., 2016), and one of the most important in industry (Ampatzoglou et al., 2016a). For simplicity, in

the rest of the chapter, when we refer to TD, we imply code TD. For assessing TD Principal and TD Interest we rely on the FITTED interest theory.

In economics, TD *Interest* and TD *Principal* are related through the interest rate: interest is calculated as a percentage of a loan (principal), paid to the lender periodically for the privilege of using that money. However, in the technical debt literature, the term interest rate has not and cannot be defined: according to Schmid (2013), it is not possible to relate principal directly to an interest rate for a given interest period. That is because the actual interest rate depends on the specific maintenance activities performed and these cannot be determined a priori. To the best of our knowledge, there is currently no approach to relate *TD Principal* and *Interest*. Clarifying the relation between principal and interest would further validate the strength of the TD metaphor in software development and maintenance. More importantly, this relation can be practically used in estimating TD indices, but also managing individual TD items. For example, if a certain aspect of TD Principal, such as a specific type of code smell, incurs more interest compared to others, then it should be ranked higher through: *preventing* the associated code smells, *prioritizing* the refactorings of those smells, and eventually *repaying* them with refactoring applications.

According to Eisenberg (2012), Lockheed Martin is monitoring TD, by using an excel sheet, in which class names are colored based on their perceived levels of technical debt. Such an approach serves two purposes: (a) ranking of classes with respect to their levels of TD, and (b) grouping classes into groups of similar TD. In this study, we build upon the rationale of such an approach for TD monitoring, by investigating if *artifacts with similar levels of TD Principal have similar levels of TD Interest*. More specifically, we investigate if there is a relation: (a) between *TD Principal* and *TD Interest*; and (b) between the aspects of *TD Principal* and *TD Interest*. To achieve the aforementioned goals and by taking into consideration the multifaceted nature of the examined concepts, we employ the Mantel test (Mantel, 1967). The main benefit of using the Mantel test, compared to a traditional correlation analysis, is that it offers the opportunity to study the relationship of concepts that can be decomposed into aspects (in our case TD principal and interest), in a hierarchical manner. Since this study focuses on exploring the relations between TD concepts and their aspects, we believe that the Mantel test is more appropriate than traditional correlation, which would be able to accurately answer only goal (a). More details for the Mantel test are provided in Chapter 6.4.4.1. In addition, we have performed *Linear Mixed Equation* (LME) modelling to investigate the extent to which the obtained results are not affected by the random effect of project selection, but they are due to the examined factors and parameters; this supports the generalizability of the results. The main findings of the study validate the relation between TD Principal and TD Interest (illustrating that classes with similar levels

of TD Principal tend to have similar levels of TD Interest), and that certain quality properties (e.g., coupling and size) should be prioritized while ranking refactoring opportunities. The rest of the chapter is organized as follows: Chapter 6.2 provides an overview of related work, in Chapter 6.3, we present the background information that is required for understanding underlying concepts. In Chapter 6.4, the case study design is overviewed. The results are presented in Chapter 6.5, and discussed in Chapter 6.6. Threats to validity are presented in Chapter 6.7.

6.2 Related Work

The goal of this chapter is to present: (a) works aiming to connect TD Principal to TD Interest—see Chapter 6.2.1; and (b) studies that focus on the quantification of TD Interest—see Chapter 6.2.2. We note that we do not discuss works on calculating TD Principal, since it is considered a straightforward task: According to Alves et al. (2016), the principal is related to the effort / cost to eliminate the debt from a given system or artifact. Current software analysis tools offer estimates of *TD Principal* based on counts of detectable violations (e.g., SonarQube, CAST, Squore).

6.2.1 Relation between TD Principal and Interest

In the literature, we have identified five studies that aim at exploring the relation between TD Principal and Interest. Table 6.2.1 outlines the studies, by presenting the TD Principal / Interest assessors, and the main conclusion of each study. Next, we present these studies in detail, and compare them against our study.

Table 6.2.1: TD Principal – TD Interest Relation

Ref.	TD Principal Assessor	TD Interest Assessor	Main Outcome
(Conejero et al., 2018)	Modularity anomalies as index of TD	Maintainability (expressed as stability) as a main characteristic related to TD interest	Improvement of modularity is related to important benefits in terms of stability, and lead to the reduction of TD interest
(Kazman et al., 2015)	Architecture roots (flawed structures)	Coupling and Cohesion (indicating higher maintenance effort)	TD items incur high maintenance penalties
(Kosti et al., 2017)	SQALE method	Structural proxies (quality metrics)	Some quality metrics are positively related and others are negatively related to TD principal
(MacCormack and Sturtevant, 2016)	Coupling between components	Defect-related activity	Highly-coupled components are more prone to defects, and costlier to maintain
(Zazworka et al., 2014)	Modularity Violations	Defect Proneness Change Proneness	Classes with more modularity violations and code smells, are more defect- and change-prone

Zazworka et al. (2014), compare the similarities and differences among four approaches for TD identification, namely modularity violations, grime buildup, code smells and automatic static analysis. Given the fact that there are plenty of tools that can automatically detect a

number of source code anomalies, the study considers four main techniques for TD detection, selected primarily by the criterion of authors' previous experience. The study aims at investigating if these approaches result in pointing out the same set of problematic issues. Moreover, the authors explore the extent to which the four techniques point to instances with high *TD Interest*. Since interest (i.e., the probable future cost of not fixing the debt) is regarded as difficult to detect and measure, the authors select to use two interest indicators, i.e., defect-proneness and change-proneness. The selection of the proxies has been made among a number of interest indicators, based on their correlation with problematic code manifestation and future maintenance cost. The authors conduct a case study, where they implement the four techniques on Apache Hadoop software. Their results show that: (a) different techniques identify different TD issues, (b) classes identified with high TD, with the use of modularity violations and code smells, seem to be more defect-prone, while modularity violations are strongly related to change-prone classes. Zazworka et al. (2014) use four different approaches to TD identification and focus on defect-proneness and change-proneness as TD interest indicators, as their goal is the comparison among the different methodologies. In our work, as mentioned earlier, we opt to focus on one TD principal estimation methodology, while we propose a more detailed approach of TD interest estimation, based on well-established maintainability metrics.

Moreover, Conejero et al. (2018) perform an empirical study aiming to evaluate if modularity anomalies at requirements level affect maintainability attributes and therefore increase system's interest. The study is based on a previously established framework that uses modularity metrics to identify and quantify modularity anomalies. A set of metrics is also used to assess the system's stability, as a particular maintainability attribute. The authors regard the presence of crosscutting concerns in a system as a negative effect on modularity, which in turn is considered as a significant index of TD (Alves et al., 2016). Modularity violations are expected to raise interest, given the fact that they may affect different quality attributes and hence cause negative impact on the system's quality. The paper attempts to prove the relation between modularity anomalies at the early stages of software development, i.e., the requirements level, with software maintainability—with regard to stability. Since maintainability is deemed a main contributor to *TD Interest*, if the aforementioned relation is proven, then modularity violations can be linked to interest escalation. The study is performed on three different industrial applications. The results suggest the existence of a relationship between the Degree of Crosscutting properties, as a measure of modularity, and the stability of a certain feature. Therefore, the authors conclude that improvement of modularity could deliver important benefits in terms of stability, enhancing the maintainability of the system and lead to the reduction of *TD Interest*. Compared to our study, Conejero et al.

(2018) focus on modularity anomalies, as indicator of its quality, while they do not explicitly refer to principal. Also, they refer to stability as a maintainability attribute, whereas our work considers: inheritance, coupling, cohesion, complexity and size, as maintainability-related properties.

Additionally, Kosti et al. (2017), explore the correlation between: (a) structural proxies (i.e., quality metrics); and (b) monetized values using the SQALE method, by conducting a case study on a set of 20 OSS projects. The results of the study suggest that a model of seven structural metrics, quantifying different aspects of quality (i.e., coupling, cohesion, complexity, size, and inheritance) can accurately estimate TD principal as appraised by SonarQube. More specifically, two coupling metrics (MPC and MOA) and one cohesion metric (LCOM) are positively related to technical debt principal, while CIS, NOP and DIT are negatively related to the accumulation of TD. While both the study of Kosti et al. (2017) and ours use the same maintainability metrics, Kosti et al. do not explicitly refer to a connection between these metrics and TD interest.

McCormack and Sturtevant (2016) address architectural debt by attempting to assess the potential value derived from a refactoring effort. To this end, they try to evaluate the relationship between system design decisions, which lead to TD, and increased costs of maintenance, which represent the interest to be paid. More specifically, the authors analyze how components with higher levels of coupling can be associated with higher maintenance costs. Firstly, based on their prior work, the authors apply a technique, namely Design Structure Matrix to analyze the system's architecture, capture the dependencies and calculate the coupling between components. This allows them to divide the components into groups, according to their levels of coupling, and then to characterize the system as a Core-Periphery one—if it possesses a large, dominant, cyclic group of components, namely the Core—or as a Hierarchical one—if it has small cyclic groups or no cyclic groups at all. In the empirical study, the authors analyze two large projects of similar size, one of them classified as a Core-Periphery system and the second as a Hierarchical one. The descriptive analysis supports the hypothesis that, for both systems, high-coupling components are more probable to experience defects and hence to be related with higher defect-related activity. The authors also perform a predictive analysis, by creating two statistical models, one to predict the probability of a component to experience a defect and another one to predict the volume of defect-related activity. Number of lines of code in a file and its cyclomatic complexity are used as control variables and are proven to be positive and statistically significant for both systems and both models. However, models with predictor variables added show a strong relation between components with high coupling and: (a) the likelihood of experiencing a defect, as well as (b) the defect-related activity. The authors also introduce a

financial analysis, where they calculate a cost per line of code per year, by component category, to maintain each system. The analysis indicates that tightly coupled components cost more to maintain than loosely coupled ones. Finally, the study suggests that repaying architectural TD by reducing coupling and lowering maintenance costs may be valuable; however, managers should also consider the cost of these refactorings. In comparison to our study, the work of McCormack and Sturtevant (2016) uses architectural design flaws that lead to tightly-coupled components as a TD proxy, and assumes that maintenance costs represent TD interest; our work focus on source code issues to identify TD principal and adopt a set of maintainability metrics as TD interest proxies, so as to provide a more thorough analysis of the relationship between the two terms.

Furthermore, Kazman et al. (2015) focus on architectural TD in terms of flawed architecture structures, termed as architecture roots (Xiao et al., 2016): these are considered technical debt items that incur high maintenance penalties as identified through coupling and cohesion. They validate their methodology by conducting a case study with an industrial partner. However, Kazman et al. (2015) do not differentiate between *TD Interest* and *TD principal*, and therefore it is not evident if they consider maintenance penalties as principal or interest, since the term debt is used collectively in that study.

Compared to related work, our study has a clear focus on identifying the relationship between TD Principal and TD Interest. To this end, it adopts well-validated valuation approaches, and provides an in-depth statistical analysis of the aforementioned relationship, based on the Mantel test and LME.

6.2.2 Quantification of TD Interest

In this chapter we present papers that deal with TD Interest calculation. In Table 6.2.2, we summarize studies that quantify TD interest, including those already presented in Chapter 6.2.1. In particular, in Table 6.2.2, we note how TD interest is assessed, if the study acknowledges the relation of TD Interest to maintainability, if it uses historical data (from software repositories) for TD interest calculation and whether they propose explicit metrics for assessing interest. We remind that maintainability and the use of historical data to calculate maintenance effort are the two core aspects of TD interest.

Table 6.2.2: TD Interest Assessment

Ref.	TD Interest Assessor	Main- tainabil- ity	History	Metrics
(Besker et al., 2017)	The extra effort needed to maintain a system due to the accumulation of TD in terms of wasted software development time	x		

Ref.	TD Interest Assessor	Main- tainabil- ity	History	Metrics
(Conejero et al., 2018)	Maintainability (expressed in terms of stability) as a main characteristic related to TD interest	x	x	
(Kazman et al., 2015)	Coupling and Cohesion indicate higher maintenance effort		x	
(Kosti et al., 2017)	Structural proxies (i.e., quality metrics)		x	x
(MacCormack and Sturtevant, 2016)	Defect related activity	x	x	x
(Nugroho et al., 2011)	Effort spent on maintenance activities based on historical data Effort needed to rebuild a system Level of software quality	x	x	x
(Zazworka et al., 2014)	Defect Proneness and Change-proneness	x	x	

Concerning the studies already discussed in Chapter 6.2.1, Conejero et al. (2018) acknowledge that “*maintainability is one of the main characteristics contributing to Technical Debt interest*”, and decide to capture it through the quality property of stability which is one of the most important maintainability attributes. Specifically, Conejero et al. (2018) measure stability by calculating the number of use cases changed in each release of the three systems they study. They define as a change in a use case: (i) a modification of the feature that the use case addresses, or (ii) a modification, addition, or deletion in the system that affects the particular feature. Similarly, Zazworka et al. (2014) recognize that defect-proneness and change-proneness—the two proxies they use for TD Interest—are connected to future maintenance costs. The authors use historical data on the number of times a class is involved in fixing bugs, that were injected, resolved, or alive in a version, to measure defect-proneness. They also use data on the number of changes affecting the class divided by the total number of changes in the repository, to calculate change-proneness.

Moreover, McCormack and Sturtevant (2016), as mentioned earlier, aim at analyzing the relationship between design decisions and maintenance costs, generally deemed to represent TD Interest. The authors approach maintenance costs through maintenance effort, i.e., effort spent on defect related activities: a metric computed through bug-tracking and version control systems represents the development activity that aims at defect correction. On the other hand, Kazman et al. (2015) focus on the relationship between architecture roots, i.e., flawed structures, and the penalties they incur in terms of higher maintenance costs. They retrieve data concerning the number of defects fixed, the number of changes associated with architecture roots that were fixed, and the number of lines of code committed to fix

the defects and to make the changes during the prior year. They then estimate the cost of refactorings to calculate the penalty in terms of maintenance cost. However, as mentioned before, Kazman et al. (2015) do not refer to this as TD interest. Finally, Kosti et al. (2017) have used structural quality metrics, that are used to assess maintainability (coupling, cohesion, complexity, inheritance, and size metrics), as proxies of TD interest.

In addition to the five studies of Chapter 6.2.1, we have identified two more studies that attempt to estimate interest and relate it to maintainability. The first one is by Nugroho et al. (2011), who use SIG/TUV's software quality assessment method for TD measurement. Particularly, they perform source code analysis that involves metrics, such as lines of code (LOC), code duplication, McCabe's cyclomatic complexity, parameter counts, and dependency counts, to map the system's quality properties to a five-star rating system, and calculate the Repair Effort, i.e., the effort needed to reach the ideal quality level, which represents the amount of the system's TD. With regard to the system's *TD Interest*, which is defined as "*the extra maintenance cost spent for not achieving the ideal quality level*", the authors estimate it as a function of: (a) the effort spent on maintenance activities within a year, calculated based on historical data, as a percentage of number of LOC estimated to change yearly for maintenance reasons; (b) an estimate of the effort needed to rebuild a system using a particular technology, determined by the total size of the system (measured in LOC or Function Points) and a language productivity factor; (c) a factor used to account for the level of quality, assuming that the higher the quality level, the less the maintenance effort.

Besker et al. (2017), attempted to quantify *TD Interest* in terms of wasted software development time through an empirical study, based on the admission that interest is defined as the extra effort needed to maintain a system due to the accumulation of TD. They perform a web-based survey answered by 258 software stakeholders and they conduct interviews consisted of unstructured, semi-structured and fully structured questions with development teams within seven software companies. The results of the study show that, according to the respondents, 36% of all software development time is on average wasted because of paying the interest of TD. The study also reveals that the system age influences the wasted time, however there is no linear correlation between the two. Moreover, Architectural Design and Requirement TD seem to cause the most negative effect on software development, whereas there is no significant differentiation on how a stakeholder's role affects the perception of wasted time. Finally, the respondents estimate that a significant percentage of wasted time is spent on understanding and measuring TD issues.

The state of the art in quantifying TD Interest is based on maintainability and change proneness (either due to defects or new features). Furthermore, the metrics used as proxies for TD Interest are either structural ones, or rely on historical data. We adopt the same strategy and use both types of metrics.

6.3 Background Information

In this section, we present background information. Chapter 6.3.1 describes the estimation of *TD Principal* using SonarQube (based on SQALE), while Chapter 6.3.2 presents our approach for assessing *TD Interest*. We note that although this information has already been presented, in this chapter we update it, since it is necessary for the semantics of this chapter.

6.3.1 TD Principal Calculation

For the purpose of this study, we have decided to estimate principal at the source code level, based on the computations provided by a widely used platform, namely SonarQube (Ampatzoglou et al., 2015a), (Li et al., 2015). SonarQube can assess the quality of software relying on quality measures and issues, such as coding rule violations. The platform algorithm was originally based upon an adopted version of the SQALE method proposed by Letouzey and Ilkiewicz (2012), in which a remediation index is obtained for the requirements of an applicable Quality Model. For example, for a requirement stating that all files should have at least 70% code coverage, the corresponding remediation action is to write additional tests; a remediation function maps effort to each action. Finally, for each artifact, the remediation index relating to all the characteristics of the model is obtained by adding all remediation indices linked to all quality requirements. The resulting SQALE Index is considered to represent *TD Principal* of the source code.

Table 6.3.1: Default metrics-tags provided by SonarQube

Aspect TD Principal	Code Smell Tags	Description	Aspect TD Principal	Code Smell Tags	Description
Understandability	brain-overload	There is too much to keep in your head at one time	Security / Runtime	Cwe	Relates to a rule in the Common Weakness Enumeration.
	confusing	Will take maintainers longer to understand than is really justified by what the code actually does		Bug	Something is wrong and it will probably affect production
Poorly Written Code	clumsy	Extra steps are used to accomplish something that could be done more clearly and concisely		owasp-.*	Relates to rules in the OWASP Top-10 security standards
	bad-practice	The code likely works as designed, but the way it was designed is wide-		unpredictable	The code may work fine under current conditions, but may fail

		ly recognized as being a bad idea			erratically if conditions change
	design	There is something questionable about the design of the code		Suspicious	It's not guaranteed that this is a bug, but it looks suspiciously like one
	lock-in	Use of environment-specific features		Security	Relates to applications' security
	unused	Unused code		Pitfall	Nothing is wrong yet, but something could go wrong in the future
Coding Standards	Cert	Relates to rules in a CERT standard	Coding Standards	Misra	Relates to rules in MISRA standards
	Convention	Coding convention violation		sans-top25-.*	Relates to the SANS Top 25 Coding Errors, which are security-related

SonarQube calculates TD Principal by identifying code smells (as the corresponding Quality Model requirements) and calculating their remediation. For identifying the existence of *code smells*, SonarQube version 7.9 (for Java) relies on 562 rules (e.g., “*Method overrides should not change contracts*”, “*Package declaration should match source file directory*”, “*Parameters should be passed in the correct order*”, “*Unused labels should be removed*”). SonarQube rules are associated with (by default) nineteen tags (see Table 6.3.1¹¹); however, the user is given the chance to create custom tags at will. Since the number of tags is quite high, it is expected to lead to a sparse table in the data collection phase, we decided to group the tags into 4 categories. We call these categories **Aspects of TD Principal**: (a) *Understandability*, (b) *Poorly Written Code*, (c) *Security/Runtime*, and (d) *Coding Standards* (see Table 6.3.1). We note that some tags could potentially belong to different tag categories, as well as, additional tag categories could have been created. For instance, groups could have also been created by inspecting the effects of a specific code smell, instead of the cause of the smell (e.g., *clumsy code* is a poorly written code, which has a negative effect on understandability). However, in our study we have selected not to consider cause-effect relations, in the sense that such a taxonomy would not uniquely categorize all tags. Thus, we have built the classification schema only based on the root of the smell and not on the affected categories. To eliminate (as much as possible) the objectivity of this categorization, each one of the senior researchers of the study proposed a classification of his own, and after some negotiation rounds (similar to the Delphi technique) they have reached a consensus on the Aspects of TD Principal, as well as the mapping of Code Smell Tags to them. Nevertheless, since this decision is still objective a relevant threat to construct validity has been identified. Moreover, we acknowledge that SonarQube is not a perfect solution for measuring TD principal, as a perfect solution does not exist; we expand on the limita-

¹¹ From the nineteen default tags of SonarQube, we deleted user-experience, since it does not relate to software maintainability.

tions of using SonarQube in the Threats to Validity Chapter.

6.3.2 TD Interest Calculation

In this study, we calculate interest using the FITTED framework, as it has been proposed in Chapter 4 and in our previous work (Chatzigeorgiou et al., 2015) and empirically validated in Chapter 5, as well as in our previous work (Tsintzira et al., 2019). In this work (Tsintzira et al., 2019), the validation was performed in an industrial setting and contrasted the scores for TD Principal and TD Interest with the perception of software engineers. The results suggested a rank correlation for 0.83 for TD Principal and 0.73 for TD Interest. We only recap the basic notions of the FITTED framework here and refer to the aforementioned works for further details.

Assuming that a system has an *actual* implementation, and a *hypothetical* optimal implementation (in terms of maintainability—i.e., ease to maintain), maintaining the optimal system would require less effort than maintaining the actual system (see Figure 1a, in Chapter 1.2). Despite the fact that a system can by no means be characterized as globally optimal, based solely on the optimization of some structural characteristics, there is a plethora of studies aiming at software optimization, guided by the application of software refactorings (e.g., (Harman, 2007), (O’ Keeffe and Cinnéide, 2008), (Ouni et al., 2016))¹². As shown in Figure 1a, in Chapter 1.2, adding a new feature A to the optimal system would need a certain effort, noted as $\text{Effort}_{(\text{optimum})}$, whereas adding the same feature to the actual system necessitates a larger effort, noted as $\text{Effort}_{(\text{actual})}$. ***The difference between these two efforts represents the TD Interest that is accumulated during this maintenance activity.***

According to FITTED (Chatzigeorgiou et al., 2015), maintenance effort is inversely related to the maintainability of the system—see (9). Although the relation between effort and maintainability is not necessarily (or by definition) linear, several studies model maintenance effort (through regression modeling) as a polynomial of maintainability indicators (e.g., (van Kotten and Grey, 2006), (Zhou and Leung, 2007)), achieving satisfactory prediction accuracy. In particular, van Kotten and Grey (2006) propose a linear model¹³, whereas Zhou and Leung (2007) propose a multivariate adaptive regression spline model¹⁴. Despite the differences in the modelling of the solutions, both studies suggest that there can be a linear relation between maintenance effort and maintainability.

¹² The relevant research area is termed search-based software engineering.

¹³ $\text{effort} = c_0 + \sum w_i \text{maintainability_indicator}_i$

¹⁴ $\text{effort} = c_0 + \sum w_i \Pi(\text{maintainability_indicator}_i - t_i)$

$$Effort = a \frac{1}{maintainability} \quad (9)$$

Given Eq. (1), the maintenance effort for the optimal system (which is unknown), can be estimated as the product of the maintenance effort for the actual system and the ratio of the maintainability of the actual over the maintainability of the optimal system (we call this ratio *Maintainability Level*) (Chatzigeorgiou et al., 2015) —see (10).

$$\frac{Effort_{optimum}}{Effort_{actual}} = \frac{\frac{a}{maintainability_{optimum}}}{\frac{a}{maintainability_{actual}}} = \frac{maintainability_{actual}}{maintainability_{optimum}} = MaintainabilityLevel \quad (10)$$

Finally, based on its definition in Figure 1, TD Interest can be calculated using the difference between the actual and the optimal effort, as follows (Chatzigeorgiou et al., 2015) —see (11):

$$\begin{aligned} TD\ interest &= \Delta Effort = Effort(actual) - Effort(optimum) \\ &= Effort(actual) \\ &\quad - Effort(actual) \times (MaintainabilityLevel) \\ &= Effort(actual) \times (1 - MaintainabilityLevel) \end{aligned} \quad (11)$$

In practice, the above calculation is multiplied by the constant value of Unit Cost of Maintenance (e.g. \$ or € per hour), but that is irrelevant for this study (although it is calculated), since the Mantel test employed in this study considers distance matrices. Therefore, as **aspects of TD Interest**, we consider (a) the Maintainability Level; and (b) the actual Maintenance Effort of the system under study. The former is related to structural properties of the corresponding system while the latter can be estimated using historical information, as elaborated below.

Maintainability Level. Although no single function can capture all aspects of quality, for the sake of simplicity, we assume that the optimal system is the one that optimizes a certain fitness function assessing the quality of software (e.g., in terms of complexity, cohesion, coupling). Thus, to calculate the maintainability level, we first identify a set of similar artifacts (e.g., classes, packages, systems—see (Ampatzoglou et al., 2018)), we then calculate the optimal value of the metric score within the set of similar (in terms of lines of code, number of methods, cognitive complexity, etc.) artifacts. The maintainability optimal artifact is an artificial one that is assigned the “*best*” metric scores, among the similar artifacts: i.e., the metric score of lowest complexity, highest cohesion, lowest coupling, etc. For example, given five similar artifacts with complexity scores: 2, 5, 3, 8, 11; the artificial optimal artifact would be assigned a complexity score of 2. Then we calculate the average ratio

of the metric score of the artifact under study, compared to the optimal value. Maintainability, although not associated to a universally accepted definition, is widely accepted as the ease of making changes into a system (ISO, 2001). The set of metrics that we have selected to use in our study for quantifying maintainability (see Table 6.3.2) belong to well-known metric suites and (Chidamber et al., 1998), (Li and Henry, 1993). The metrics selection was based on a secondary study by Riaz et al. (2009), which reported on a systematic literature review (SLR) aimed at summarizing software metrics that can be used as maintainability predictors.

In particular, Riaz et al. (2009) have performed a quality assessment of maintainability models, through a quantitative checklist, in order to identify studies of high-quality score, i.e., studies that provide reliable evidence. More specifically, the checklist was comprised of 19 questions and each model was assessed for each criterion by a three-point scale: yes, no, or partially, with associated scores of 1, 0, and 0.5 respectively. The range of the total score of each study was between 0 and 19. All studies that have scored 7 or below were excluded from the list of selected studies, whereas among the studies with the highest scores were those of van Kotten and Gray (2006), and Zhou and Leung (2007). Both studies (i.e., (van Kotten and Gray, 2006) and (Zhou and Leung, 2007)) have used the same definition of maintainability and they have been based on two metric suites proposed by Li and Henry (1993) and Chidamber et al. (1998), i.e., two well-known object-oriented set of metrics. The employed suites contain metrics that can be calculated at the source-code level, and can be used to assess well-known quality properties, such as inheritance, coupling, cohesion, complexity and size. We note that according to Riaz et al. (2009) another study (performed by Misra (2005)) scored equally to the previously mentioned ones. However, Misra was using metrics coming from multiple suites (2 out of the 4 are already considered), and we preferred to select those that were common in the studies.

Table 6.3.2: Maintainability Properties and Metrics

Property	Metric	Description
Inheritance (Inh)	DIT	Depth of Inheritance Tree: Inheritance level number, 0 for the root class.
	NOCC	Number of Children Classes: Number of direct sub-classes that the class has.
Coupling (Cpl)	MPC	Message Passing Coupling: Number of send statements defined in the class.
	RFC	Response for a Class: Number of local methods plus the number of methods called by class methods.
	DAC	Data Abstraction Coupling: Number of abstract types defined in the class.
Cohesion (Coh)	LCOM1	Lack of Cohesion of Methods: Number of disjoint sets of methods in the class.

Complexity (Com)	CC	Cyclomatic Complexity: Average cyclomatic complexity of methods in the class.
	WMPC	Weighted Method per Class: Weighted sum of methods. Each method of the class is assigned to a weight equal to 1.
Size (Size)	SIZE1	Lines of Code: Number of semicolons in the class.
	SIZE2	Number of Properties: Number of attributes and methods in the class

Maintenance Effort: Since the evolution of the software cannot be predicted, it is not possible to foresee what kind of modifications will be made in a system during future releases. Hence, we base our assessment of future maintenance effort on historical data, by considering past effort spent on maintenance activities. More specifically, as maintenance effort we assume the average lines of code added/deleted/modified between all pairs of successive versions of a system. This strategy has been used in a variety of studies e.g., (Conejero et al., 2018), (Kazman et al., 2015), (Kosti et al., 2017), (MacCormack and Sturtevant, 2016), (Nugroho et al., 2011), and (Zazworka et al., 2014).

6.4 Empirical Methodology

Case study is an observational method that is used for studying phenomena in a real-life context (Runeson et al., 2012). This case study has been designed and is presented according to the guidelines of Runeson et al. (2012).

6.4.1 Research Objectives and Research Questions.

The goal of the study, as mentioned in Chapter 6.1, is to investigate the relation between *TD Principal* and *Interest*, as well as the relation between the *Aspects of TD Principal* (see Chapter 6.3.1) and the *Aspects of TD Interest* (see Chapter 6.3.2). An overview of the aspects of *TD Principal* and *TD Interest*, is presented in Figure 6.4.1. We note that the calculation of each aspect at a higher level, is an aggregation of the lower level:

- We remind that ***TD Principal*** has four aspects that correspond to the categories of code smell tags. As explained in Chapter 6.3.1 *TD Principal* is calculated as the sum of *TD Principal* due to Understandability, Security/Runtime Issues, *TD Principal* due to Poorly Written Code, and *TD Principal* due to the Violation of Coding Standards.
- ***TD Interest*** on the other hand has two aspects (at the second level): Maintainability Level (ratio of maintainability of actual vs. optimal case) and Maintenance effort. The aggregation formula from the 2nd to the 1st level is provided in Chapter 6.3.2—see Eq. (11). Maintainability is further decomposed into five structural properties, while Maintenance effort constitutes historical change. We consider these five structural properties and the historical changes as aspects of *TD interest* at the third level. The function for aggregating the 3rd level aspects to Maintainability Level is the average of

the distance from optimal, as explained in Chapter 6.3.2.

Figure 6.4.1: Aspects of TD Principal and TD Interest

Based on this overview, and the goal of this study (i.e., to explore the relation between TD Principal and TD Interest, as well as their aspects), we have formulated three research questions.

RQ₁: *Is TD Principal related with TD Interest?*

This research question aims to explore if source code artifacts with a similar level of TD Principal are presenting similar levels of TD Interest. The existence of such a relation, would suggest that interest-related information could be subsumed by principal-related information, and therefore TD management, based only on *TD Principal* would make sense. With respect to Figure 6.4.1, this research question explores the relation of the 1st level of *TD Principal* with the 1st level of the *TD Interest* hierarchy.

RQ₂: *Which Aspects of TD Interest are more related to TD Principal?*

Given the fact that *TD Interest* calculation considers two 2nd level aspects (maintainability level and maintenance effort), we first explore if any one of these two *aspects* of *TD Interest* is more related to *TD Principal* (**RQ_{2.1}**). The answer to this RQ can shed light into the importance of the constituents of TD Interest, and guide researchers in their future attempts to quantify interest. Next, we focus on the maintainability level aspect of *TD Interest* and

investigate which of the 3rd level aspects (i.e., coupling, cohesion, complexity, size, or inheritance) are more related to *TD Principal* (**RQ_{2.2}**). We note that we are not performing a similar analysis for the historical change data (i.e., we do not further split this concept), since no actionable outcome can be reached by such an investigation: you cannot change the history of a project. In contrast, structural properties that appear to be of more interest can be improved or prioritized; for instance, in case cohesion ends up being an important property, classes that suffer from low cohesion can be refactored (e.g., split method, split class). With respect to Figure 6.4.1, this research question focuses on the relation between the 1st level of *TD Principal* and the 2nd and 3rd Level of *TD Interest*.

RQ₃: *Which Aspects of TD Principal are more related to TD Interest?*

The third research question deals with comparing different *Aspects* of *TD Principal* (of the 2nd level), with respect to the interest that they are expected to incur. In this research question, we investigate if and which of the *Aspects* of *TD Principal* are related to the highest interest. This question becomes extremely relevant for preventing, prioritizing, and repaying specific aspects of *TD Principal* within the same technical debt item. For example, if it turns out that understandability issues are producing more interest compared to poorly written code issues, then the issues of the aspect should be prioritized. With respect to Figure 6.4.1, this research question focuses on the 2nd level of *TD Principal* and the 1st level of *TD Interest*.

6.4.2 Case Selection and Unit Analysis

This study is characterized as multiple, embedded case study (Runeson et al., 2012), in which the cases are the OSS projects and the units of analysis are their classes. The reason for using OSS systems is the vast amount of available data in OSS repositories, in terms of versions and classes. To obtain data from high-quality projects (see Table 6.4.2) that evolve over a period of time, we require that the software systems:

- Are popular OSS project of the Apache community. This ensures that the investigated projects are recognized as important by the OSS community, i.e., there is substantial system functionality and substantial development activity in terms of bug-fixing and adding requirements.
- Are written in Java. We include this criterion because of the employed metric calculation tools.
- Contain more than 70 classes. This ensures that we will not include “toy examples” in our dataset. After data collection, a manual inspection of the projects has been performed to guarantee that the classes are not trivial.
- Have more than 5000 commits. We have included this for similar reasons to the first

criterion. Although the selected number of versions is ad/hoc, it is set to a relatively high value, in order to guarantee high activity and evolution of the project. Also, this number of revisions provides an adequate set of repeated measures as input to the statistical analysis.

Table 6.4.2: OSS Projects Selected for the Case Study

Name	Short Description	#classes
Apache XML Graphics (XMLGraph)	Apache XML Graphics Commons is a library that consists of several reusable components used by Apache Batik and Apache FOP	109
Commons Math (ComMath)	It is a library for mathematics and statistics components	901
Commons Collection (ComCollection)	The Apache Commons Collections package contains types that extend and augment the Java Collections Framework.	307
Commons Net (ComNet)	Apache Commons Net library implements the client side of many basic Internet protocols.	148
Commons IO (ComIO)	Commons IO is a library to assist with developing IO functionality.	113
Commons Jelly (ComJelly)	Jelly is a tool for turning XML into executable code.	73
Http Components – Core (HTTPCore)	Http Core is a set of HTTP transport components, used to build client and server services.	368
Http Components – Client (HTTPClient)	Http Components is responsible for creating and maintaining a toolset of Java components.	294
Apache Struts (Struts)	Struts is an MVC framework for creating modern Java web applications.	622
Xerces 2 Java (Xerces2Java)	Xerces2 is a library for parsing, validating and manipulating XML files	665

6.4.3 Data Collection and Pre-Processing

For the quantification of TD Principal, we used the SonarQube API to obtain the TD Principal metric for each one of the classes of the projects under analysis. Next, for each row in the dataset we recorded the number of instances of issues in each Tag Category, which are concentrated in the class (5 variables). Regarding TD Interest, we used the TDM toolkit¹⁵ of the SDK4ED platform¹⁶, developed in the context of the SDK4ED project¹⁷. On the completion of data collection, each class (unit of analysis) was characterized by 12 variables—10 maintainability metrics, maintenance history, and TD Interest. The pre-processing was completed by the deletion of rows that contained missing values. Missing values can be found in cases a maintainability metric cannot be calculated: e.g., CC cannot be calcu-

¹⁵ <https://github.com/AngelikiTsintzira/Technical-Debt-Management-Toolbox>

¹⁶ <http://sdk4ed.se.uom.gr/>

¹⁷ <https://sdk4ed.eu/>

lated for abstract methods, or LCOM cannot be calculated for interfaces.

6.4.4 Data Analysis Methodology

Since the main objective of the study is to examine whether classes that present similar levels of TD Principal, also produce a similar amount of TD Interest, classical approaches such as correlation analysis are not able to provide straightforward answers. Although traditional correlation coefficients can be used to explore the nature and strength of the pairwise relationship between variables of a multivariate dataset, they can only assess which subset of software metrics is associated to the TD Principal at the lowest level of the quality hierarchy. The current approach focuses on the similarity (dissimilarity) of classes to examine the association between TD Principal and TD Interest, by considering that *TD Principal* and *TD Interest* are multifaceted concepts that can be assessed through various aspects (see Chapter 6.3 and Figure 6.4.1) synthesizing in turn, multidimensional spaces of metrics. Through this approach, we believe that researchers and practitioners will be able to acquire significant knowledge in a more straightforward and intuitive manner, since the interpretation of the results resembles the way of human decision-making by comparing similar cases (i.e., classes), as in case-based reasoning (CBR) process.

6.4.4.1 Mantel Test

To this regard, we make use of a multivariate statistical methodology, namely the Mantel test (Mantel, 1967) that has been extensively applied in many scientific areas such as: health, ecology, biology, population genetics. The rationale of the approach is to evaluate the association between the corresponding positions of all pairs of observations from two dissimilarity matrices computed by either univariate or multivariate data. The procedure is further augmented with a randomization mechanism based on the permutation of the rows and columns of one of the two dissimilarity matrices, to test the null hypothesis that the two dissimilarity matrices are uncorrelated. We have also to point out that the Mantel test has been used in software engineering: (a) to evaluate the difference between perspectives in the determination of the relative importance of impact analysis issues of software (Rovegård et al., 2008); and (b) for the calibration of the analogy-based software cost estimation model (Keung et al., 2008). An outline of the approach that we have used for applying the Mantel test, in our study, is described below:

1. Each analysis element (i.e., box in Figure 6.4.1) is represented by either a vector or a matrix, in which the rows correspond to the n classes of each project and columns comprise the metrics representing a specific element (see Table 6.4.4.1.a). A simple concept is represented by a column vector (Table 6.4.4.1.a), e.g. TD Principal, the measurements can be represented in the following form $TD_{Principal}^T =$

$(TD_{Principal_{c_1}}, \dots, TD_{Principal_{c_n}})^T$. Regarding multifaceted concepts, e.g. the quality property of *Size* evaluated by two metrics (*Lines of Code* and *Number of Properties*, 3rd Level, Table 6.4.4.1.a), the measurements can be compiled in the form of a tabular matrix

$$\begin{bmatrix} SIZE1_{c_1} & SIZE2_{c_1} \\ \vdots & \vdots \\ SIZE1_{c_n} & SIZE2_{c_n} \end{bmatrix}$$

where $SIZE1_{c_i}$ and $SIZE2_{c_i}$ represent the measurements of i class regarding the two metrics *Lines of Code* and *Number of Properties*, respectively. The followed approach provides us the ability to capture a relation between sets of metrics that quantify two concepts that can be either simple or multifaceted; whereas traditional univariate correlation analysis would be able to indicate the relationship between pairs of scalar metrics.

Table 6.4.4.1.a Analysis Element Representation

Level	Concept	Element	Representation	Metrics
1	TD Principal	TD Principal	Vector	Total TD Principal
1	TD Interest	TD Interest	Vector	Total TD Interest
2	TD Principal	Types of TD Principal Issues	Matrix	TD Principal due to Understandability Issues TD Principal due to Security/Runtime Issues TD Principal due to Poorly Written Code TD Principal due to the Violation of Coding Standards
2	TD Interest	Maintainability Level	Matrix	Coupling, Cohesion, Complexity, Size Inheritance
2	TD Interest	Maintenance Effort	Vector	Average Number of Lines Changed between Commits
3	TD Interest	Coupling	Matrix	Message Passing Coupling Response for a Class Data Abstraction Coupling
3	TD Interest	Cohesion	Vector	Lack of Cohesion of Methods
3	TD Interest	Complexity	Matrix	Cyclomatic Complexity, Weighted Method per Class
3	TD Interest	Size	Matrix	Lines of Code, Number of Properties
3	TD Interest	Inheritance	Matrix	Depth of Inheritance Tree, Number of Children Classes

- From each analysis element we calculate a distance matrix for each pair (i, j) of n classes for a given project—see example in Table 6.4.4.1.b. For the case of *TD Principal* and *TD Interest* (first levels of hierarchies), the first step of the method involves the evaluation of two distance matrices, $Distance(TD_{principal})$ and $Distance(TD_{interest})$ representing the distances between the pair of classes (i, j) of the two vectors (1st Level, Table 6.4.4.1.a). In the case of a multifaceted TD concept (e.g. quality property of *Size*,

3rd Level, Table 6.4.4.1.a), the distance matrix is evaluated based on the measurements of all the associated metrics (see Table 6.4.4.1.a). We note that dissimilarities between each pair of classes are evaluated on the standardized measurements ([0,1]) to be immune to metrics' range. For example, the investigation of the relation between *TD Principal* (1st Level) and the element of *Size*, described by two metrics (*Lines of Code* and *Number of Properties*) is based on distance matrices evaluated through the following formulae:

$$a_{ij} = \sqrt{\left(TD_{Principal_{c_i}} - TD_{Principal_{c_j}}\right)^2} \quad (12)$$

$$b_{ij} = \sqrt{\left(SIZE1_{c_i} - SIZE1_{c_j}\right)^2 + \left(SIZE2_{c_i} - SIZE2_{c_j}\right)^2} \quad (13)$$

Table 6.4.4.1.b: Distance-Matrix Example for TD Principal (1st Level) and Size Element of TD Interest

Distance (TD _{Principal})		Classes				Distance (TD _{Interest-Size})		Classes			
		C_1	C_2	...	C_n			C_1	C_2	...	C_n
Classes	C_1	0	α_{12}	...	α_{1n}	Classes	C_1	0	b_{12}	...	b_{1n}
	C_2	α_{21}	0	...	α_{2n}		C_2	b_{21}	0	...	b_{2n}
	...	...	...	...	...		...	...	...	...	...
	C_n	α_{n1}	...	...	0		C_n	b_{n1}	...	...	0

3. Then, we take separately for each project all possible combinations of analysis elements (i.e., pairs of boxes: one from the *TD Principal* and another from *TD Interest* hierarchies), and calculate the *Mantel's r correlation coefficient* between the corresponding matrices.
 - a. for **RQ₁**, we use the vectors of TD Principal and TD Interest of the first level
 - b. for **RQ_{2.1}**, we use the vector of TD Principal, the matrix of Maintainability Level and the vector of Maintenance Effort
 - c. for **RQ_{2.2}**, we use the vector of TD Principal, the matrices of Coupling, Cohesion (the only vector), Complexity, Size, and Inheritance
 - d. for **RQ₃**, we use the vector of TD Interest and the matrix of Types of TD Principal Issues

We have also to note that due to the symmetrical nature of a distance matrix, only the elements from the upper or lower triangle matrix are used.

6.4.4.2 Statistical Inferential Process

After the discovery of similarity patterns, the next critical issue is to investigate whether the observed phenomena can be generalized to the population of OSS projects. Given the fact

that the case study is characterized as multiple (i.e., using many projects as subjects—see Chapter 6.4.2), there is an imperative need to adopt an appropriate statistical inferential mechanism in order to derive conclusions regarding the population of OSS projects with similar characteristics. Towards this direction, an aggregated dataset comprising the Mantel’s correlation coefficients evaluated from the classes of each OSS project for paired analysis elements is constructed. For example, for the case of **RQ₂**, the dataset can be expressed via a long-format matrix as presented in Table 6.4.4.2. As we can observe, each row of the matrix comprises the Mantel coefficient evaluated from the distance matrices of *TD Principal* and the alternative two aspects of *TD Interest* (Maintainability Level and Maintenance Effort) for a given project of our experimental setup. The question is to examine the potential effects of aspects of *TD Interest* on *TD Principal*.

For this reason, we use the *Linear Mixed Effects* (LME) modelling statistical technique (Pinheiro and Bates, 2006) that is able to model simultaneously two types of effects that are (a) the fixed and (b) the random effects. In the terminology of LME models, the term “*fixed effect*” is used to depict factors influencing the mean value of a response variable, whereas a “*random effect*” (i.e. *Project* in our experimental setup) may have an impact on the variance of the response variable. The reason to control the project as a random effect is that our dataset represents OSS projects that are selected from an infinite unknown population of projects.

Regarding **RQ_{2.1}**, our aim is to examine whether there is a difference between the 2nd Level Aspects of *TD Interest* on how they are related with the *TD Principal*. We essentially investigate which of the two aspects correlates better with *TD Principal*. This difference can be formally modeled as an effect of the factor “*TD Interest Aspects*” (with two discrete levels, namely *Maintainability Level* and *Maintenance Effort*) on the Mantel correlation in the presence of the project’s random effect. So, we actually consider the Mantel *r* as response variable while the type of aspect (fixed effect) and the project (random effect) are the two explanatory variables (Table 6.4.4.2). As far as the *second goal* of **RQ₂** concerns, which is the investigation of whether there are 3rd Level Aspects of *TD Interest* that are more related to *TD Principal*, a similar approach is followed. In this case, the repeated measures design is represented via a matrix, where the examined effect is the 3rd Level *Aspect of TD Interest*: *Inheritance*, *Coupling*, *Cohesion*, *Complexity*, and *Size*) with the same random effect, i.e. the *Project*. Finally, a similar process has been also applied for **RQ₃**. The rest of the repeated measure designs is presented in Appendix B.

Table 6.4.4.2: Repeated Measures Design (RQ_{2.1})

Mantel <i>r</i>	TD Principal	Aspect of TD Interest	Project
r_{11}	Total TD Principal	<i>Maintenance Effort</i>	Project 1

r_{21}	Total TD Principal	<i>Maintainability Level</i>	Project 1
...	...	...	...
r_{1n}	Total TD Principal	<i>Maintenance Effort</i>	Project n
r_{2n}	Total TD Principal	<i>Maintainability Level</i>	Project n

6.5 Results

In this chapter, we provide the answers to the RQs of this study. In Table 6.5 and Figure 6.5, we present the descriptive statistics for the aspects of TD Interest (second and third level) and TD Principal (second level), respectively.

Figure 6.5: Aspects of TD Principal Frequency

Note: Aspects of the TD Principal are represented as counts of detected issues in all projects. Code Smell Tags that are not presented in Figure 6.5 (compared to Table 6.3.1) have zero instances in our dataset.

Table 6.5. Descriptive Statistics for TD Interest and its Aspects

Metric		N	M	SD	Min	Max
TD Interest		3599	59.12	114.62	0.92	1698.10
Aspect of TD Interest	Metric	N	M	SD	Min	Max
Maintenance Effort – Historical Change	LOC	3599	46.04	83.90	0.20	1401.00
Maintenance Difficulty – Inheritance (Inh)	DIT	3600	2.17	1.40	1	9
Maintenance Difficulty – Inheritance (Inh)	NOCC	3600	0.76	2.55	0	41
Maintenance Difficulty – Coupling (Cpl)	MPC	3600	41.00	106.17	0	2531
Maintenance Difficulty – Coupling (Cpl)	RFC	3600	31.68	41.43	0	532
Maintenance Difficulty – Coupling (Cpl)	DAC	3600	0.29	1.05	0	20
Maintenance Difficulty – Cohesion (Coh)	LCOM	3389	75.27	356.25	0	8759
Maintenance Difficulty – Complexity (Com)	CC	2790	1.51	1.27	1	22.57

Maintenance Difficulty – Complexity (Com)	WMPC	3600	9.17	14.45	0	276
Maintenance Difficulty – Size (Size)	SIZE1	3600	56.50	91.80	1	1361
Maintenance Difficulty – Size (Size)	SIZE2	3600	11.12	17.16	0	238

Note-1: N refers to the number of classes in which the metric was calculated (e.g., CC cannot be calculated for interfaces / abstract classes). *M*, *SD*, *Min* *Max* stand for: mean, standard deviation, minimum and maximum of the metric

Note-2: The range of values for all the metrics that represent the aspects of TD Interest is $[0, +\infty)$ and TD Interest itself is measured in euros, using as Unit Cost of Maintenance the 1.8324 dollars per line of code (see (Chatzigeorgiou et al., 2015))

6.5.1 Relation between TD Principal and Interest (RQ₁)

Mantel’s correlation coefficients (r) for each pair of *TD Principal* and *TD Interest*, as evaluated from the classes of each OSS project, are summarized in Figure 6.5.1 ranging from weak ($r_{min} = 0.271$, ComNet) to very strong ($r_{max} = 0.820$, ComIO) correlation. The y-axis of Figure 6.5.1, demarcates the regions of “no”, “weak”, “moderate”, “strong”, or “very strong” relation (Evans, 1996), whereas the x-axis does not have a specific conceptual interpretation: it is just used for spreading projects in the full-width of the graph, to facilitate readability.

Figure 6.5.1: Mantel’s coefficients between Principal and Interest

Based on the evaluation of coefficients and their corresponding p -values, we can observe that there is noted a statistically significant correlation between the distance matrices of *TD Principal* and *TD Interest* for the whole set of the examined projects implying that classes with similar levels of TD principal have similar levels of TD interest. Based on the scales presented in Figure 6.5.1, for 80% of the projects the relation between TD Principal and TD Interest is at least moderate. In the second step of the analysis, an LME model is fitted to investigate the strength of the observed association to the population of OSS projects. The parameter of the LME model fitted on the accumulated results demonstrates a mean value

of 0.540 signifying a moderate correlation between *TD Principal* and *TD Interest* aspects.

Classes with similar levels of TD Principal tend to have similar levels of TD Interest. The strength of this relation is at least moderate and statistically significant.

6.5.2 Relation between TD Principal and Aspects of TD Interest (RQ2)

Comparing 2nd-level Aspects of TD Interest (RQ2.1). Given the relation between principal and interest, we further drill down to investigate if either of the second-level aspects of interest, Maintainability Level (structure) or Maintenance Effort (history) presents a higher effect on this relation. Figure 6.5.2.a presents the distributions of the correlation coefficients between *TD Principal* and the two aspects of *TD Interest* (Maintainability Level and Maintenance Effort), in which each asterisk denotes the sample coefficient for an examined

Figure 6.5.2.a: Distributions of coefficients: TD Principal and Aspects of TD Interest (Maintainability Level / Maintenance Effort)

project of the case study. The p -values of the tests revealed statistically significant correlations for all pairwise comparisons between *TD Principal* and both aspects of *TD Interest*. Regarding the strength of the association for the Historical aspect of *TD Interest* (i.e., *Maintenance Effort*), the coefficients range, again, from weak ($r_{min} = 0.160$, for Xerces2Java) to strong ($r_{max} = 0.700$, for ComJelly), whereas the results are similar for the Structural aspect of *TD Interest* ($r_{min} = 0.230$, for ComIO, $r_{max} = 0.610$, for ComJelly).

To investigate the effect of the *Aspects of TD Interest* on the evaluated correlation coefficients, we fitted again, an LME model. The model did not reveal a statistically significant main effect of the *Aspects of TD Interest* on the examined coefficients ($F = 0.415$, $p = 0.536$) denoting that the two aspects are correlated to *TD Principal* to the same extent. The parameter estimates for the population mean values of correlation are 0.384 and 0.418 for *Maintainability Level* and *Maintenance Effort* aspects, respectively. Given the fact that both aspects of interest have merit, and do not differ significantly, it is important to investi-

gate if the aggregated measure of *TD Interest* is more related to *TD Principal* compared to the association of the two aspects (in isolation) to *TD Principal*. The results suggest that *TD Principal* seems to present a higher correlation to the aggregated *TD Interest* metric (1st level of the *TD Interest* hierarchy) compared to the aspects of *TD Interest* (second level of hierarchy).

To investigate the generalizability of the aforementioned results, we fitted an LME model incorporating the fixed effect of the factor *Hierarchy Level* (1st Level/2nd Level). Based on the results of the previous model (i.e. insignificant differences between the two *Aspects of TD Interest*), we have to clarify that the category *Second Level* aggregates the correlation coefficients evaluated from both *Maintainability Level* and *Maintenance Effort* aspects of *TD Interest*. The model revealed a statistically significant main effect of the factor *Hierarchy Level* on the mean values of the correlation coefficients ($F = 5.927, p = 0.025$).

Table 6.5.2.a: LME - Main Effect of Factor Hierarchy Level of TD Interest

	<i>b</i>	<i>SE</i>	<i>Df</i>	<i>t</i>	<i>p</i>
First Level	0.540	0.053	19	10.248	< 0.001
Second Level	-0.139	0.057	19	-2.435	0.025
Note: <i>b</i> , <i>SE</i> , <i>df</i> , <i>t</i> , <i>p</i> stands for parameter estimations, standard error of the estimates, degrees of freedom, t-statistic and p-value, respectively. The reference category for factor Hierarchy Level of TD Interest is First Level.					

Interpreting the parameter estimates of the LME model (see Table 6.5.2.a), the aggregated measure of *TD Interest* presents a higher mean correlation value to *TD Principal* compared to the two aspects in isolation. The negative sign of the parameter estimate for the *Second Level* of *TD Hierarchy* implies that the mean correlation coefficient is 0.139 lower than the corresponding value evaluated from the *First Level of TD Hierarchy* (0.540). For the rest of this study, we do not perform any analysis on the 2nd level of *TD Interest*.

The aggregated measure for TD Interest seems to be a more representative metric for capturing the divergence of classes in terms of their TD principal compared to the individual Historical or Structural metrics

Comparing 3rd-level Aspects of TD Interest (RQ_{2.2}). In this section, we investigate whether there are certain structural aspects of *TD Interest* (3rd level of the *TD Interest* hierarchy), that are more related to *TD Principal* (RQ_{2.2}). The main motivation for this, as explained in Chapter 6.2.2, is the fact that structural properties are directly linked to actionable results; e.g., a tentative importance of lack of cohesion, can underline the importance of conforming to the Single Responsibility Principle (Martin, 2003).

Figure 6.5.2.b: Distributions of Mantel's coefficients between TD Principal and Maintainability Predictors

Figure 6.5.2.b summarizes the Mantel's correlation coefficients (r) evaluated from the classes of each OSS project based on the dissimilarities for each pair of *TD Principal* and maintainability metrics (Inh: *Inheritance*, Com: *Complexity*, Coh: *Cohesion*, Cpl: *Coupling*). The results suggest that the strength is heavily dependent on the type of maintainability predictor, ranging from statistically significant very weak correlation (e.g. for *Inh* predictor, $r_{min} = 0.06$, ComMath) to strong correlation (e.g. for *Size* predictor, $r_{max} = 0.671$, HTTPClient). Generally, *Size* seems to be the most related maintainability predictor of *TD Interest* to *TD Principal*. In contrast, *Inheritance* presented low coefficients that are statistically significant for 4 out of 10 cases.

The findings of the LME for the accumulated results present a statistically significant coefficient, demonstrating a significant main effect of the maintainability predictors on the response variable, $F = 13.061$ and $p < 0.001$. The parameter estimates of the model (Table 6.5.2.b) reveal that the mean value of correlation between *TD Principal* and *Inheritance* (reference category of the FITTED model) is very weak 0.061 ($r_{Inh} = b_{Intercept} = 0.061$) and significantly lower than the corresponding population mean values of *Coupling* ($r_{Cpl} = b_{Intercept} + b_{Cpl} = 0.061 + 0.381 = 0.442$), *Cohesion* ($r_{Coh} = b_{Intercept} + b_{Coh} = 0.061 + 0.297 = 0.359$), *Complexity* ($r_{Com} = b_{Intercept} + b_{Com} = 0.061 + 0.306 = 0.367$), and *Size* ($r_{Size} = b_{Intercept} + b_{Size} = 0.061 + 0.439 = 0.500$).

Table 6.5.2.b LME - Main Effect of Aspect of TD Interest

	<i>b</i>	<i>SE</i>	<i>df</i>	<i>t</i>	<i>p</i>
Intercept	0.061	0.062	30	0.988	0.331
Complexity	0.306	0.063	30	4.844	< 0.001
Cohesion	0.297	0.063	30	4.705	< 0.001
Coupling	0.381	0.063	30	6.021	< 0.001
Size	0.439	0.063	30	6.947	< 0.001

Note: *b*, *SE*, *df*, *t*, *p* stands for parameter estimations, standard error of the estimates, degrees of freedom, t-statistic and p-value, respectively. The reference category for factor maintainability predictor of TD Interest is Inh

Figure 6.5.2.c Post-hoc analysis for LME model (main effect of Maintainability Predictors of TD Interest)

The post-hoc analysis through Tukey's HSD test signifies statistically significant differences ($p < 0.001$) in all pairs between *Inheritance* and the other four quality properties (Figure 6.5.2c—cases in which the error bar does not cross the vertical dashed line on 0.0). Regarding the *Size*, there is a statistically significant difference with *Complexity* ($p = 0.030$) and *Cohesion* ($p = 0.017$), but no difference compared to *Coupling* ($p = 0.702$). Therefore, we consider the strength of the relation of *Coupling* to *TD Principal*, similar to the strength of the relation between *Size* and *TD Principal*. Finally, the rest pairwise comparisons do not indicate statistically significant differences.

Size and Coupling are the maintainability-related properties that are most closely related to TD Principal, followed by Cohesion and Complexity.

6.5.3 Relation between Aspects of TD Principal and TD Interest (RQ₃)

In this section, we present the results on the relation between aspects of TD Principal and TD Interest (RQ₃). We remind that the aspects of TD Principal correspond to the four tag categories (Understandability, Poorly Written Code, Security/Runtime, and Coding Stand-

ards), that group the nineteen tags of code smells (see Chapter 6.3.1). Subsequently we evaluate the correlation of those four aspects to the first level of the *TD Interest Hierarchy*. *TD Principal* aspects present statistically significant correlations to *TD Interest* for the majority of the examined projects (37 out of 40 cases—see Figure 6.5.3) ranging from weak ($r_{min} = 0.137$, ComCollections) to strong ($r_{min} = 0.699$, ComJelly). Regarding the LME model incorporating the factor aspects of TD Principal, the findings did not reveal a statistically significant main effect on the examined coefficients ($F = 1.126$, $p = 0.356$).

Figure 6.5.3: Distributions of Mantel's coefficients between Tag Categories of code smells and TD Interest

All four categories of code smells (aspects of TD Principal) present a moderate relation to TD Interest.

6.6 Discussion

In this chapter, we discuss the main findings of this study, first interpreting the obtained results, and then providing useful implications for researchers and practitioners.

6.6.1 Interpretation of results.

The goal of this study is to explore the relation between *TD Principal* and *Interest*, as well as the relation between the aspects of both concepts. First, we have been able to provide empirical evidence on the existence of a relation between principal and interest. Therefore, although no causal effect can be assumed between principal and interest, we have provided the first well-based indications on the existence of a relation between interest and principal: classes with similar levels of TD Principal tend to produce similar levels of TD Interest. We note that identifying the form of the relationship (linear or any other type) would require a

different kind of analysis. Thus, the existence of a relation similar to the one of economics still needs investigation, probably through a different study setup that can assess causality. We have also unveiled a relation between TD Principal and 4 (out of 5) 3rd level aspects of *TD Interest* (namely: size, coupling, cohesion, and complexity). This relation appears to be stronger for size and coupling, and less strong for cohesion and complexity:

- **Size:** The relation between TD Principal and size (i.e., classes of similar size tend to have similar TD Principal) is intuitive in the sense that the more lines of code exist in the system, the more rules are expected to be violated. However, in most cases, this is not an actionable result as refactoring only for the sake of reducing size is very rarely done. In contrast, the size of software presents a linear growth over time; thus, it is of paramount importance that new code inserted into the system has as few rule violations (TD principal) as possible. Nevertheless, this observation provides two interesting implications: (a) TD Principal normalization (by size) makes sense for comparing classes of different sizes; and (b) the identification of design hotspots (in terms of TD Principal) should not be performed at a system-wide, but between similar (in terms of size) neighborhoods of classes; in the sense that artifacts of different size will not be directly comparable.
- **Coupling, Cohesion and Complexity:** The relation of TD Principal with these three quality attributes (Aspects of TD Interest—i.e., classes with similar TD Principal tend to have similar levels in these three quality properties), is indirect: we conjecture that developers who pay attention to software design (e.g., improved modularity, or low complexity) are also careful not to violate source code programming rules. The fact that coupling is more strongly related to TD Principal compared to cohesion and complexity denotes that a property reflecting the design rather than the implementation, is more important for maintenance. We remind that coupling, as calculated in this study (i.e., MPC, RFC, and DAC), can be calculated from design level artifacts (e.g., UML class or sequence diagrams); whereas complexity (CC—count of iteration and selection statements) and cohesion (LCOM—attributes used in method bodies) can only be captured by parsing source code artifacts.

Finally, regarding the specific aspects of TD Principal, the results suggest that the four aspects do not differ statistically significantly, in terms of the TD Interest that they are associated with. We would expect that TD interest (in the way that it is assessed in this study) as a structural property, is conceptually closer to two of the TD Principal aspects: code understandability and design practice violations. However, the other two TD Principal aspects (coding standards and run-time and security violations) seem equally related to TD interest. This is not intuitive especially for the relation between run-time and security violations and

TD interest; a study investigating a possible causality between the two would be particularly interesting.

6.6.2 Implications to Researchers and Practitioners.

Regarding *practitioners*, there has always been demand for an accurate calculation of interest to drive the prioritization of repaying TD. While there are relatively mature ways to calculate TD principal (mostly through source code analysis), TD interest is more elusive as it depends on knowing future changes. The establishment of a relation between TD principal and interest, implies that TD principal can be safely used for TD prioritization: paying back the TD items with the highest principal will very likely also reduce the TD interest paid in the system. In practice, prioritization of TD items is required whenever a development team receives an intractable number of refactoring suggestions from a TDM tool (which is the usual case when large rule-sets are applied on large software systems). Despite the fact that practitioners could either prioritize based on TD Principal or TD Interest; in fact, they have long been prioritizing TD items with large principal. Our results provide empirical evidence that this is a sound practice, since the amount of *TD Principal* is related to the amount of *TD Interest*, in the sense that *classes with similar levels of TD Principal tend to have similar levels of TD Interest*. However, this observation does not downgrade the significance of TD Interest assessment: we have also provided evidence that especially for TD repayment, emphasis should be placed on improving specific quality properties (i.e., coupling and size), which have proven to be linked to the concentration of more TD Principal. Additionally, based on the findings of this study, some interesting future work opportunities have been identified. Therefore, we encourage *researchers* to:

- **Study of causality.** The establishment of a correlation between TD interest and principal and their various aspects, begs the question whether causality also exists between them. This is especially interesting for third-level aspects of interest and second level aspect of principal. Does complexity, for example, cause understandability rules violations? Does low cohesion and high coupling cause poorly written code? Such a causal study could be designed and performed through a controlled experiment, in which the researchers would control the amount of TD Principal in several variations of a system, and seek the actual maintenance time (TD Interest) for different amounts and types of TD Principal. The necessity (and difficulty) of targeting causation, instead of correlation has been discussed in detail in Dagstuhl 2014 on "Software Development Analytics" (Di Penta, 2016), (Menzies, 2016).
- **Replicate and Generalize.** The results of this study have been obtained by studying well-known and high-quality Java projects. Therefore, there is a need to replicate the case study in other languages and projects of different levels of quality, so as to ensure

the generalizability of our results. Similarly, the results need to be confirmed in different programming paradigms (other than object-oriented). We note that such a study, supposing that both extensions are made, would require the addition of two new factors (programming language and paradigm) and an inferential analysis that would target the exploration of the effect of these two factors in the identified relations

- ***Extend the concepts of TD Principal and TD Interest to other types of TD.*** By considering that this study limits the calculation of TD to code, we believe that an interesting extension would be towards other types of TD, such as architecture, requirements or documentation TD. For instance, requirements debt could concern costs such as the delay of developing features, whereas architecture debt could involve the impossibility to evolve the system, or the effect on other quality attributes or even on the social aspects of the organization. Nevertheless, given the level of abstraction of these concepts, we see this research work more qualitative (e.g. involving experts) than quantitative.

6.7 Threats to Validity

In this chapter, we discuss potential threats to the validity of our case study: construct validity, reliability, and external validity (Runeson et al., 2012). Internal validity is not considered, since causal relations are not in its scope.

6.7.1 Construct Validity

Construct validity is related to the way in which the selected phenomena are observed and measured. In this study the investigated concepts are *TD Principal* and *TD Interest*. On the one hand *TD Principal* is quantified through SonarQube. SonarQube is the most frequently used tool for measuring *TD Principal* (Ampatzoglou et al., 2015a), (Li et al., 2015), in the sense that is the most widely used in research and practice. Although SonarQube is an established tool, it focuses on code TD, neglecting other types of TD, like architecture debt, requirements debt, etc. According to Martini et al. currently in industry static analyzers (such as SonarQube) are used to analyze the source code in search of TD. Only in few cases out of the respondents in their survey (15 companies) practitioners built their own metrics tools for checking (language-specific) rules or patterns that can warn the developers of the presence of TD (Martini et al., 2018). In a similar discussion, Yli-Huumo et al. (2016) discuss SonarQube as the mostly used tool for TDM in the eight development teams that they have involved in their case study. Despite the identified limitations, especially in the level of Architectural Technical Debt (ATD), SonarQube is considered as extremely useful for code TD identification, monitoring, measurement and prioritization. Additionally, although SonarQube could be configured to provide more accurate results (e.g., change reme-

diation times), such a practice is not prominent in the literature, where researchers do not perform any re-configuration of the tool (García-Valls et al., 2019), and (Schnappinger et al., 2019).

On the other hand, in the literature there is no established way to measure *TD Interest*. This is due to the fact that an accurate measurement of interest would require the simultaneous maintenance of two software solutions: an optimal and an actual one, and the anticipation of future maintenance activities. Besides the inability to forecast future changes, such an approach is unrealistic for two reasons: (a) there is no way to define a universally accepted optimal system, and (b) it is cost inefficient to maintain two real systems just aiming to accurately measure technical debt interest. According to industrial practitioners, acknowledge that there are no indicators that show the amount of interest paid or predicted if the refactoring. Research prototype tools for interest assessment are not employed in practice yet and should be integrated to provide overall indicators to provide help to the stakeholders to estimate and prioritize TD. Thus, the TD research community shall intensify their work on introducing such tools and indicators (Martini et al., 2018). Therefore, as the current state-of-the-art stands *TD Interest* can only be assessed through proxies. In this study, we selected metrics that assess maintainability as a proxy of interest. More specifically, we selected ten object-oriented metrics (grouped in 5 categories/aspects of *TD Interest*) measured at source code, although, in literature, maintainability has been linked to various metrics. Metrics' selection was based on empirical evidence in the literature suggesting that a combination of these metrics is the optimal maintainability predictor (Riaz et al., 2009). The model for synthesizing the aforementioned values in a unified value for TD Interest relies on solid mathematical calculations, given the assumption that maintenance effort is inversely proportional (linearly) to maintainability. This assumption, although it cannot be validated without a controlled experiment, relies on previous studies (van Koten and Gray, 2006), (Zhou and Leung, 2007) and is considered as.

Additionally, we need to note that both *TD Principal* and *TD Interest* are measured at source code level. However, TD is a wider concept that represents inefficiencies at the whole software development lifecycle, and therefore the source code analysis is not comprehensively studying the phenomenon. Thus, our results are not representative of TD holistically as a phenomenon, but only of a subset of it. Nevertheless, code TD is the most studied type of technical debt in the state-of-research (Alves et al., 2016) and one of the most important in the industry (Ampatzoglou et al., 2016a). Finally, with respect to RQ₃, we note that the results heavily rely on the classification schema that we have proposed for Aspects of TD Principal. Although the schema has not been validated and relies on the expert opinion of the senior researchers of this study, it is developed in a systematic way.

Thus, also given the fact that such an endeavor could not be conclusive, we believe that it serves the goal of this study, since it is explicitly presented and acknowledges all of its inherent limitations.

6.7.2 Reliability

With regard to reliability, we consider any possible researchers' bias, during the data collection and data analysis process. The design of the study, concerning data collection, does not contain threats, since all data are automatically extracted by tools, without any subjective configuration. Moreover, with respect to the data analysis process, to mitigate any potential threats to reliability, three researchers were involved in the process, aiming at double checking the work performed and thus reducing the chances of reliability threats. Furthermore, the detailed case study protocol presented in Chapter 6.4 enables the repetition of the study, as well as the provision of a replication package¹⁸. However, we need to note that the clustering of code smells under specific tag categories is subjective and could have been differently performed. Nevertheless, we believe that the clustering is intuitive and forms a well-justified decision.

6.7.3 External Validity

Concerning external validity, a potential threat to generalization is the possibility that performing the study on different projects of different languages might affect the retrieved correlations. However, we believe that the selected projects, given their size and complexity, represent a realistic real-world system. Additionally, the results of the study are not applicable to non-object-oriented systems, in the sense that *TD Interest* in such systems could not be assessed through properties such as inheritance, coupling and cohesion, which are applicable only in OO software modules. Finally, we note that since the interpretation of the results is based solely on the understanding of the authors on the TD concepts, and not through an additional qualitative study with industrial stakeholders, they cannot be generalized to an industrial context, without additional validation.

6.8 Conclusion

This study aims to investigate the interrelation between *TD Principal* and *TD Interest* from two perspectives: (a) to understand the underlying relations between the two concepts, and (b) to provide a way for efficient TD management. To achieve these goals, we have performed a case study on 3600 classes retrieved from 10 Apache projects. The concepts of *TD Principal* and *TD Interest* have been decomposed to multiple aspects that assess differ-

¹⁸ <https://se.uom.gr/wp-content/uploads/interest-principal-empirical-relation/replication.zip>

ent views of the concepts. Given the hierarchical structure of the concepts (*TD Principal* and *TD Interest*) the Mantel test has been used for the examination of their relation and Linear Mixed Effects models for assessing the generalizability of the obtained results. The results of the analysis suggested that *TD Principal* is related to *TD Interest*, and that *TD Principal* is more closely related to the interest aspects of size and coupling, followed by cohesion and complexity. Regarding *TD Principal* aspects, the one that appears to be more strongly interrelated to higher levels of interest is code smells, whereas by further focusing on code smells, we have collected evidence that smells that hinder source code understandability are the ones that are more urgent to resolve in the sense that they are related to higher levels of *TD Interest*. Given the aforementioned outcomes, various implications for researchers and practitioners have been drawn. In particular, regarding practitioners we have suggested a strategy for technical debt prevention, repayment, and prioritization, based on technical debt interest amount.

Based on: Ar. Ampatzoglou, E. M. Arvanitou, A. Ampatzoglou, P. Avgeriou, A. A. Tsintzira, and A. Chatzigeorgiou — “*Architectural Decision-making as a Financial Investment: An Industrial Case Study*”, Information and Software Technology, Elsevier, 2020.

Chapter 7 – Architectural Decision-making as a Financial Investment: An Industrial Case Study

Making architectural decisions is a crucial task but also very difficult, considering the scope of the decisions and their impact on quality attributes. To make matters worse, architectural decisions need to combine both technical and business factors, which are very dissimilar by nature. We provide a cost-benefit approach and supporting tooling that treats architectural decisions as financial investments by: (a) combining both technical and business factors; and (b) transforming the involved factors into currency, allowing their uniform aggregation. Apart from illustrating the method, we validate both the proposed approach and the tool, in terms of fitness for purpose, usability, and potential limitations. To validate the approach, we have performed a case study in a software development company, in the domain of low-energy embedded systems. We employed triangulation in the data collection phase of the case study, by performing interviews, focus groups, an observational session, and questionnaires. The results of the study suggested that the proposed approach: (a) provides a structured process for systematizing decision-making; (b) enables the involvement of multiple stakeholders, distributing the decision-making responsibility to more knowledgeable people; (c) uses monetized representations that are important for assessing decisions in a unified manner; and (d) enables decision reuse and documentation. The results of the study suggest that architectural decision-making can benefit from treating this activity as a financial investment. The various benefits that have been identified from mixing financial and technological aspects are well-accepted from industrial stakeholders.

7.1 Motivation

One of the most important tasks of software architects is to make **architectural decisions** (Kruchten, 2008). This is a particularly challenging task since architecture decisions often affect large parts of the system and impact quality attributes; consequently, one needs to get them right from the beginning as they are very hard to change in the future (Manteuffel et al., 2016). The difficulty of these decisions is further aggravated by having to consider not only technical factors (such as reusability, maintainability, testability), but also market and business factors (such as time-to-market, achieved market share). However, it is often hard,

if not impossible to weigh these different factors against each other, e.g., comparing the impact of a decision on coupling and cohesion with the impact on time-to-market. Being forced to compare “apples and oranges”, architects often make ad-hoc, intuitive judgement calls, which are prone to bias (van Vliet and Tang, 2016).

In this study, we aim at addressing the issue of taking heterogeneous factors into account and comparing them against each other when making architectural decisions. Specifically, we propose an approach, named ADMIT, that looks at this problem from a *financial* perspective, by treating decisions as *investments* through a *cost-benefit analysis*. This is achieved by expressing all inputs in monetary terms, and then comparing the financial benefits obtained by applying the architectural decision, against its costs. The main benefits from treating architectural decisions as financial investments are: (a) the ability to mix and aggregate technical and business factors of the decisions, e.g., improvement in market share, cost to write code, and effort required in future maintenance activities; and (b) the power to express all factors in a uniform way, especially using monetary terms, which are more easily perceived by higher management than purely technical ones.

Figure 7.1: Overview of the Proposed Approach

The proposed approach relies on defining a set of costs and benefits related to the architectural decision under study, and comparing them to assess if the decision should be taken or not. To aid software architects with the selection of costs and benefits, the proposed approach comes with a set of default cost and benefit models (see Figure 7.1). Specifically, the *modelling of costs* relies on the well-established PAF (Prevention-Appraisal-Failure) cost model—see Chapter 7.3.1 and Chapter 7.3.2; whereas, the *modelling of benefits* relies on the architectural quality business goals, as defined by Kazman et al. (2001) (see Chapter 7.3.3). In addition to these default models, a software architect is of course free to introduce custom models, depending on the relevant technical and business factors at play. We note that some costs or benefits (especially quality-related ones) might be difficult to monetize; for such cases, we expect the decision-maker to provide (at best) an informed estimate.

Subsequently, the models are instantiated, i.e., estimated values are assigned to each factor. As already mentioned, all cost and benefit models provide their outcome as a currency value, allowing them to be aggregated into a total cost and benefit. To ease the applicability of the ADMIT approach, we have developed an accompanying tool. The tool provides a level of automation but the approach is not fully automated: in practice, a number of costs and benefits are evaluated by experts.

We note that ADMIT is not a full-fledged and standalone architectural decision-making approach, in the sense that it only considers individual decision alternatives, while it leaves out the typical elements of the decision (e.g., design issue, alternatives, relations with other decisions, decision status) – see for example the work of van Heesch et al. (2012) or Kruchten et al. (2006). Instead, ADMIT *supports cost-benefit assessment of individual decisions* and can be used in combination with other architectural decision-making approaches and tools (it is not meant as a replacement for them). For example, each of the alternative decisions can be assessed with respect to cost-benefit with ADMIT, and then the optimal one can be further assessed through another decision-making process that considers the interplay between decisions. Furthermore, the proposed approach cannot (realistically) be applied to all decisions, due to the effort required to build cost/benefit models. Therefore, it is expected to be useful only for the most important decisions, especially high-risk or high-impact decisions (Poort and van Vliet, 2012).

To validate the applicability of the proposed approach and also evaluate its accuracy and usability, the approach has been applied in a real-world industrial case from the low-energy embedded systems domain, and has been empirically evaluated by practitioners. In particular, a case study was designed and executed, in which four practitioners discussed their experiences in architectural decision-making, identified benefits and limitations on their current way of working, applied the proposed approach in the context of their day-to-day job, and evaluated its perceived usefulness and usability. To make the study as unbiased as possible, we have employed method triangulation, using four data collection methods (namely: interviews, focus groups, questionnaires, and task analysis). The results have been analyzed using well-accepted methods for qualitative studies (Spencer, 2009).

The rest of the chapter is organized as follows: in Chapter 7.2, we present related work on architectural decision-making. In Chapter 7.3, we provide background information that is necessary for the better understanding of the study. In Chapter 7.4, we present the proposed approach and the accompanying tool, which we validate in Chapters 7.5 (study design) and 7.6 (results). The obtained results are discussed in Chapter 7.7: (a) with respect to existing literature; (b) future work opportunities; and (c) implications for researchers and practitioners.

7.2 Related Work

In this chapter, we present related work that has been published in the field of architectural decision-making, both in terms of methods that employ cost-benefit analysis, and in terms of tool support. Note that, since we were not able to identify tools that support decision-making based specifically on cost-benefit analysis, we provide a more generic discussion on tool support for architectural decision-making. We note that the main advancements compared to existing studies are discussed at the end of this section, and not per paper.

Cost-Benefit and Risk Analysis for Architectural Decision-Making. Kazman et al. (2001) propose the use of Cost-Benefit Analysis Method (CBAM), which is applied for assessing candidate solutions to an architectural problem. In particular, the authors suggest that based on the business goals of the software, several architectural decisions can be evaluated. As a first step for applying the process, an estimate of the cost of applying each solution is made by the software architect. Second, the architect selects the quality attributes of interest and specifies the anticipated quality improvement, through a concept termed utility. Finally, these quality gains are transformed to benefits, which can be compared to the cost of each alternative solution. The main difference between CBAM and our approach, is that the former uses scenarios to calculate benefits while it does not elaborate on the calculation of costs; our approach instead focuses on cost estimation using PAF and provides default cost and benefit models. Also, the use of a unified currency unit is arguably more effective when interacting with business stakeholders. In the same line of research Lee et al. (2009), adopt the Analytic Hierarchy Process (AHP) and the Analytic Network Process (ANP) methods to perform an elaborate cost-benefit analysis of architectural decisions. The proposed process consists of two phases: the development of a Scenario-based Architectural Strategy, and the application of the Cost Benefit Analysis to assess the existing strategies (i.e., design alternatives). Both processes (AHP and ANP) receive input from various stakeholders, so as to initially calculate the best, worst, current and desired values for the decision parameters. Next, based on the aforementioned input, each strategy is assigned with a weight, through pair-wise comparisons. Upon the calculation of costs and benefits, the Return of Investment (ROI) for each strategy is calculated and used for prioritization. Finally, Poort and Van Vliet (2012) suggest that architecting can be treated as a risk and cost management discipline. The risk and cost are mapped to core architectural practices and principles, which in turn can be evaluated, based on their cost and risk values. Nevertheless, we need to note that the two aspects are not synthesized, and that most of the emphasis is given to risk, whereas our study focuses on decision costs and benefits. Such a point of view can help decision-makers to identify the key concerns that drive decisions, through a simple and objective way. The use of risk and costs (similarly to our approach)

enables the communication with business stakeholders, who are used to these concepts, when performing decision-making. The proposed approach, namely Risk-and Cost Driven Architecture (RCDA) has been empirically validated through a survey on more than 150 architects. The results of the validation suggest that RCDA has a significant positive impact on their architecting work.

Tool Support for Architectural Decisions. Kruchten et al. (2005) argue in favor of an explicit representation of design decisions and other factors that drive those decisions, as a means to facilitate a system's evolution. They presented an initial set of use cases on the documentation of Architectural Knowledge (AK), a general term that mostly focuses on design decisions and their rationale. They also stressed the importance of an efficient visualization of the ontology of design decisions.

In the work of Babar and Gorton (2007), the authors develop PAKME, a web-based tool for Managing Software Architecture Knowledge, supporting services of storing, retrieving, and updating artifacts of architectural knowledge, as well as the generation of reports that represent the relationships between different architectural artifacts, explicate them or depict their positive or negative effects on each other. Capilla et al. (2008) present their own web-based application, namely the ADDSS—Architecture Design Decision Support System, which captures and documents architectural design decisions. The tool also explicitly links these decisions, so it is possible to estimate the effect of adding, modifying, or removing a decision. SAW (Software Architecture Warehouse), proposed by Nowak and Pautasso (2013) is an architectural decision management tool that aims at supporting co-located and distributed design teams, under the perspective of collaborative decision-making. SAW allows the architect to follow relevant architectural issues and appropriate solutions for those issues. Tang et al. (2007) suggest AREL (Architecture Rationale and Elements Linkage), an architecture model that captures the relationship between Architecture Rationale (AR) and Architecture Elements (AE). The authors implement this model as an add-in for Enterprise Architect which supports architects in the design traceability and reasoning process. The Knowledge Architect tool suite, proposed by Liang et al. (2010), aims at supporting collaborative architecting. To this end, the tool is based on a Knowledge Repository, as a central location for storing and retrieving. The tool suite has been validated in two industrial case studies and is considered to simplify Architectural Knowledge capturing and sharing.

All the aforementioned tools are analyzed in two subsequent studies alongside the state of practice in tool support for Architectural Knowledge (AK) management (Tang et al., 2010) (Capilla et al., 2008). The first of these studies provides a framework for the comparison of these tools, according to criteria based on literature and architecture activities (Tang et al.,

2010). The second study is an extension of the first several years later, proposing new taxonomies (based on tool features) and aiming at exploring new trends and developments by conducting an industrial case study (Capilla et al., 2008). However, to date, these tools have remained research prototypes and have not been adopted in industry.

More recently, Manteuffel et al. (2016), have developed a decision documentation tool, namely Decision Architect, which is meant to fulfil the expectations of industry and that software architects would find easy to adopt. The validation of the tool has been performed through two industrial case studies. The results of this work show that architects are generally positive on adding a documentation tool in their daily routine and perceived the tool as useful and relevant. Additionally, Li et al. (2014) in their book chapter on software economics suggest that the costs and benefits of architectural decisions (aiming at reducing technical debt) should be specified as an integral part of the decision documentation process. However, in that study the authors do not describe any process on how these costs and benefits should be retrieved, or synthesized.

Based on the above, our work extends the body of knowledge on architectural decision-making, by: (a) providing a set of default cost and benefit models to guide architects in more systematically selecting cost and benefits; (b) aggregating the results of more than one costs and benefits directly in monetary values, addressing the issues raised from the use of different units when assessing different quality attributes; (c) providing tool-support that guides the architect throughout the cost-benefit analysis.

7.3 Background Information

In this chapter we present background information for understanding the underlying concepts of the proposed approach (costs and benefits). In particular, in Chapter 7.3.1, we present the Prevention, Appraisal, and Failure model that is used for modeling the *costs* of the approach. In Chapter 7.3.2, we provide a review on the uses of PAF cost models in software; these are used for building the default cost models of the proposed approach. Finally, in Chapter 7.3.3 we present a list of business goals in the domain of software engineering, which act as a basis and inspiration for the default *benefit* models presented in Chapter 7.4.

7.3.1 Prevention, Appraisal and Failure Cost Model

Quality cost models have been developed in the 1950s (Kim and Nakhai, 2008) as a means to serve the purpose of Total Quality Management (TQM) (Porter and Rayner, 1992). Feigenbaum used the term Total Quality Control (TQC) (Feigenbaum, 1956), and defined it as “*an effective system for integrating the quality-development, quality maintenance, and quality-improvement efforts of the various groups in an organization so as to enable pro-*

duction and service at the most economical levels which allow for full customer satisfaction". The term "Quality Costs" lacks a broadly accepted definition, as it may be conceived either as referring to the costs of improving quality (quality costs) or the costs incurred due to low quality (poor quality costs) (Kim and Nakhai, 2008). To address this issue, Juran proposed the economic conformance level (ECL) model in his 1951 Quality Control Handbook (Juran, 1951), which suggested the terms of conformance and non-conformance costs. Conformance costs refer to actions taken to ensure the production of high-quality goods, while non-conformance costs apply when resources are spent because the product does not conform to specifications. The Prevention – Appraisal – Failure (PAF) model, was introduced by Feigenbaum (1961), and classified conformance costs into prevention and appraisal costs, while non-conformance costs were divided into internal and external failure costs (Kim and Nakhai, 2008).

- **Prevention costs** include costs that prevent or reduce the risk of defective products – e.g., process control, product and service design and redesign, supplier relations, audit and screening, preventive maintenance.
- **Appraisal costs** are incurred to evaluate the product's conformance to specifications – e.g., raw material inspection, in-process inspection, quality audit.
- **Internal Failure costs** are incurred by defective products, before the product reaches the consumer – e.g., scrap, rework, reinspection of products.
- **External Failure costs** are those incurred by defective products, discovered after the product is delivered to the customer – e.g., warranty charges, complaint handling, and lost sales.

In the PAF model, while prevention and appraisal costs are increasing, failure costs tend to decrease, as more faults are revealed at an early stage. In any case, it is more costly to modify a defective product as it proceeds in the production phases. Total quality cost is defined by adding prevention, appraisal and failure costs, while the optimal quality level is the one minimizing total quality costs, i.e., the investment to quality is exactly the one required to achieve the least failure costs.

Figure 7.3.1: The Prevention, Appraisal and Failure Cost Model (Peimbert-García et al., 2012)

Figure 7.3.1 depicts the relationship between prevention-appraisal and failure costs, as well as the optimal (in terms of cost) level of quality. Although the PAF model has many drawbacks and limitations (Kim and Nakhai, 2008), (Porter and Rayner, 1992), it is the most widely used (Kim and Nakhai, 2008), (Porter and Rayner, 1992) and “*almost universally*” accepted quality cost model. From Figure 7.3.1, it becomes clear there is a specific point (in the conjunction of the two lines—Failure Costs and Cost of Prevention plus Appraisal) in which the Total Cost of Quality (CoQ) get minimized.

7.3.2 Prevention, Appraisal and Failure in Software Cost Assessment

In order to derive the default models for our approach, we conducted a meta-analysis on how the Prevention-Appraisal-Failure Cost Model has been applied in software engineering. The main goal of the meta-analysis was the identification of the most prominent types of costs related to the software quality assurance process. The meta-analysis used the primary studies of a systematic literature review (Karg et al., 2011). More specifically, the author of the thesis, with the second and third co-authors of the article, have gone through the primary studies to identify the most prominent costs, based on their frequency. In total, we looked at 87 studies from Karg et al. (2011) that are related to prevention, appraisal, or failure costs. Within these studies, we identified 79 prevention, 114 appraisal, and 136 failure costs; these however were not unique, i.e., the same costs appeared more than once. Next, we went through the costs, so as to consolidate them and merge similar ones into a single title; this merging was driven by the fact that we would not be able to empirically evaluate the approximately 330 individual costs that had been identified. To achieve this, we followed the Open Card Sorting methodology, proposed by Spencer (2009) as follows: (a) we identified themes (i.e., super-categories) from the costs as identified in the primary

studies (without any processing); (b) we reviewed the themes to find candidates for merging; and (c) we defined the names of the final themes.

The final list of the consolidated costs in each category are presented in Table 7.3.2, accompanied with the frequency of primary studies, in which the cost has been referenced. For example, the “*Necessary Rework (Debugging, Quality Improvement)*”, has been mentioned in 55 studies as a “*Failure Cost*” (F). We note that Table 7.3.2 maps the Costs to the respective categories as reported by Karg et al. (2011). Therefore, there are costs presented in multiple categories: For example, “*Costs for Hiring Quality Personnel*” is reported in all three categories by the primary studies but later on in this chapter, we classify it under “*Prevention Costs*”, since it is the prevalent category, and is closer to our own interpretation of the specific cost. We have done the same for all costs that map to more than one category. As a means of confirming the list of default cost/models presented in Table 7.3.2, we have searched for similar studies (outside the field of software development), and contrasted the identified costs. We found only one similar study, conducted by Elizondo-Noriega et al. (2017); the top costs in each category in Table 7.3.2, are also found in the study of Elizondo-Noriega et al.

Table 7.3.2: Costs Identified in the Literature

Type	Specific Cost	#	Type	Specific Cost	#
Prevention	Costs for Hiring Quality Personnel	9	Approval	Cost for Performing Code Inspection	12
	Costs for Performing Meetings	9		Cost for Performing Design Reviews	9
	Costs of Training	8		Cost for Performing Unit Testing	9
	Increase Preventive Maintenance Effort	6		Cost for Using Quality Assurance Tools	6
	Cost of Producing Documentation	3		Cost for Performing Quality Audits	6
	Increase Cost for Project Planning	3		Cost for Performing Acceptance / Integration Testing	6
	Cost of Prototyping	3		Cost for Performing Metrics-Based Quality Assurance	5
	Cost for Reusing Proprietary Software	3		Cost for Performing Regression Testing	5
	Costs for Enabling the Use of Standards	3		Cost of Producing Documentation	2
	Costs for Applying Code Generation	1		Costs for Hiring Quality Personnel	2
	Costs for Lowering Programming Productivity	1		Costs for Improving the Quality Assurance Process	2
	Costs of Over-Engineering	1		Costs for Performing Glass-box Testing	1
	Costs for Applying Quality Function Deployment	1		Costs of Prototyping	1
	Costs for Improving System’s Usability	1		Costs of Simulation	1
Failure	Necessary Rework (Debugging, Quality Improvement)	55		Costs for Performing System Testing	1
	Declining Market Positioning	17		Costs for Enabling Test Automation	1
	Additional Costs for Marketing	3		Costs for Performing Usability Testing	1

Type	Specific Cost	#	Type	Specific Cost	#
	Lost Opportunity Costs	3			
	Costs Caused by Damages	2			
	Costs by Investing on Failure Recovery Research	2			
	Costs for Hiring Quality Personnel	2			
	Costs Caused by Legal Damages	2			
	Costs Caused by Non-Operational Losses	1			
	Costs of Supporting of Multiple Versions	1			

7.3.3 Business Goals and Financial Benefits

Among the studies mentioned in Karg et al. (2011), we have been able to identify only one study that mentions concrete benefits (at the business level) in software development. Although this study was considered, it was not particularly well-cited in literature or accepted as an authoritative source. Thus, we opted instead to adopt the following set of business goals as established by Bass et al. (2003), which is one of the most cited and recognized sources in the architecture community and was derived in practice through numerous industrial software development projects:

- **Growth and Continuity of the Organization.** The possible contribution that a software system can have on the sustainability of the organization, focusing on its growth and continuity.
- **Meeting Financial Objectives.** The extent to which a system can contribute to the revenue generated (if sold as standalone or a service) or saved by its use (if used for internal purposes).
- **Meeting Personal Objectives.** Individuals have various goals associated with the construction of a system, including the increase of personal reputation, knowledge, and experience gains.
- **Meeting Responsibility to Employees.** This goal refers to issues related to employees involved either in development or in operation, such as ensuring their role in development teams, giving them opportunities to evolve professionally, or provide them a safe working environment.
- **Meeting Responsibility to Society.** Some organizations see themselves as being in business to serve society; e.g., limiting resource usage, green computing, ethics, safety, open source issues, security, privacy.
- **Meeting Responsibility to State.** Organizations that develop government systems, are intended to meet responsibility to state (e.g., export controls, regulatory conformance, or supporting government initiatives).

- **Meeting Responsibility to Shareholders.** Goals complementary to the financial objectives' category. Liability protection and regulatory conformance are issues that meet the goals of this category.
- **Managing Market Position.** This goal refers to topics related to the strategy used to increase or hold market share, various types of intellectual property protection, or the time-to-market.
- **Improving Business Processes.** Through this goal stakeholders aim at improving business processes, which in turn can enable new markets, new products, or better customer support.
- **Managing Quality and Reputation of Products.** This business goal is related to product quality from the perspective of customers, such as branding, recalls, types of potential users, quality of existing products, and testing support and strategies.
- **Managing Change in Environmental Factors.** Since business context for a system might change, this goal intends to encourage the stakeholders to identify change scenarios in the business goals for a system.

7.4 Architectural Decision-making as a Financial Investment

In this chapter, we present the proposed architectural decision-making approach, namely **ADMIT** (*Architectural Decision-Making as a financial Investment*). The central idea of the approach is that each decision is treated as a financial investment; as aforementioned in Chapter 7.1, the scope of ADMIT is *cost-benefit estimation of individual design decisions*, leaving out other elements of a decision or relations between decisions. Thus, all parameters that affect an individual decision are monetized, so that they can be aggregated as total benefits or costs, with the latter comprising the costs of prevention, appraisal and failure. The proposed approach involves five steps, as presented in Figure 7.4.a. In the next paragraphs we elaborate on each of the steps of the approach. In addition, to make the steps more concrete, we demonstrate them through a running example that is executed with the help of the accompanying tool. The running example will be expanded further in Chapter 7.5; it concerns an IoT system, where a decision needs to be made to move part of a computation from the CPU to the GPU so as to save energy consumption and improve performance.

Figure 7.4.a: Architectural Decision-Making as a Financial Investment

Initialization. The decision maker (e.g., a software architect or project manager) first provides some general information on the project under study that could range from completely technical parameters, such as performance, maintainability, or reusability, to purely business ones, such as the competitiveness of the tool in the market, etc. As a first step, the user can either select the project to which the decision corresponds to, or create a new one. While creating a project, some basic characteristics of the project are being initialized, such as the Lines of Code, Market Size, Market Share.

Identification of a Decision. Next, a need for making an architectural decision (e.g., adding a new library, switching to a new framework) is considered. In this step, the user creates a new decision, or selects an already existing decision to revisit. In our running example, we revisit the decision to move part of the computations from the CPU to the GPU (see Figure 7.4.b).

Please select a decision, or create a new one

Name	Owner	Date	
Move Computations to GPU	Lazaros Papadopoulos	10/10/2019	✓

Create New Decision

Create Decision

Name of Project	Bridge Software
Description of Decision	
Owner	
Date	

Create

Figure 7.4.b: Identification of a Decision (Selection or Creation)

Attach Cost and Benefit Models to the Decision. In this step, the decision maker considers the costs and benefits that potentially arise from the decision. To facilitate the application of this step, the approach comes with some predefined cost and benefit models. As default

cost models we have used the most frequent costs of Table 7.3.2; specifically, we considered all costs that were mentioned in at least 3 primary studies (see Chapter 7.3.2). As default benefit models, we used the business goals discussed in Chapter 7.3.3. We note that the number of business models is limited to three (compared to the eleven business goals in Chapter 7.3.3), since the rest could not be straightforwardly linked (in a generic way) to monetized benefits, directly gained from the outcome of the decision. For example, consider the goal “*Meeting Responsibility to Society*”, and suppose a company that develops software that optimizes CO₂ emissions in cars. This company gets a better brand name, since they have contributed towards a ‘greener’ automotive, but this benefit can only be quantified by the increase in the market share that the company gets in other products, or new contracts. In such cases, we prompt the architect to use the default model on increased market share; however, the users of the approach can still use any other business goal by appropriately monetizing them.

Due to space limitations the complete list of default cost and benefit models is provided in Appendix C.2, which lists them along with an equation that shows the parameters of the models and the type of aggregation. The equations and aggregation types are defined based on our perception and the definition of cost / benefit models in the original study. To ensure a common and sensible perception of the cost / benefit models that lead the development of acceptable equations, the authors performed a short focus group. The focus group was comprised of the authors themselves (one of them having financial background, and three of them experience in professional software development) and 3 individuals that are currently active software engineers / architects. The outcome of the focus group was a first validated version of the models, while the accuracy and the ease of instantiating these models (i.e., predicting the values of the parameters), is subject to further validation by our industrial case study. We note that the unit for all functions are currencies (e.g., \$, or €). Finally, the fact that the number of default benefits is significantly lower than the number of default costs, is because benefits are more strongly related to specific decisions. For instance, an architectural refactoring that aims at reducing energy consumption, would lead to less energy costs for the users of the software. However, providing this very specialized benefit as a default one, might be confusing for the vast majority of decision-makers, who would not use it. One can observe from the table of Appendix C.2, that the provided models are either *time-framed* or paid/obtained *at-once*. These two types can (and must) be considered in the same model, in the sense that they are both valid for a decision. The default is to use a one-year period for the assessment of all time-framed costs/benefits so as to be able to combine them in a single cost-benefit model. If one has a different business model, one can timeframe the decision differently. The only restriction is that the same timeframe

Select an existing cost model, or add a custom one

Type	Name	Equation	
Cost Model	Acceptance and Integration Testing	Cost to Develop Test Cases + (Test Cases to be Executed * Time Needed to Execute Test * Testers Hourly Rate)	+
Cost Model	Code Inspection	(Number of Inspections per Month * 12) * Developers Hourly Rate * Number of Participants	+
Cost Model	Declining Market Positioning (e.g., poor quality, delay of release)	Lost Customers * Price	+

Create a Custom Cost Model

Model Name	Type of Equation
<input type="text"/>	Linear

Setup the Equation

Linear Equations

General Form:

$$Y = a_0 + a_1X_1 + a_2X_2 + \dots + a_nX_n$$

Special Cases:

$Y = a_0 + a_1(X_1 + X_2) + a_2X_3 + \dots + a_nX_n$
 $Y = a_0 + a_1(X_1 * X_2) + a_2X_3 + \dots + a_nX_n$

One parameter: $X_1 + X_2$

One parameter: $X_1 * X_2$

Product Equations

General Form:

$$Y = X_1 * X_2 * \dots * X_n$$

Special Cases:

$Y = (X_1 + X_2) * X_3 + \dots * X_n$
 $Y = (X_1 / X_2) * X_3 + \dots * X_n$

One operand: $X_1 + X_2$

One operand: X_1 / X_2

Parameter Name	Parameter Type	Weight in Equation or Value (if const)	
<input type="text"/>	Constant	<input type="text"/>	+

Create Custom Model

Figure 7.4.c: Create a Custom Model

needs to be applied comprehensively to all equations. We emphasize that apart from the default cost-benefit models, the decision maker is free to develop custom ones that fit the purpose of a specific decision. If the default models are used, the decision maker can change the parameters and weight of each parameter in the corresponding equation.

Tool-wise, upon the creation of the decision, the decision-maker needs to select the cost and the benefit models that correspond to that decision. For this, there are two options: (a) create a new model (see Figure 7.4.c); or (b) select a default cost or benefit model (see Figure 7.4.d).

While creating new cost / benefit models, the decision-maker first provides a name for the custom model (e.g., “*Cost of Migrating a Solution to the Cloud*”) and then selects the type of equation (i.e., linear or product). Given the nature of the equation (e.g., $Y = a_0 + a_1X_1 + a_2X_2 + \dots + a_nX_n$), the decision-maker declares the $x_1 \dots x_n$ variables and their weight for aggregation, as well as the constant value (a_0). For instance, supposing that migration to cloud costs a standard amount of 1000\$, plus 3\$ per migrated line of code, the decision-maker needs to add two parameters: (a) the standard amount—*type*: constant, *value*: 1000 and (b) the lines of code to be migrated—*type*: parameter, *weight*: 3.

Instantiate the Models. In this step, the decision-maker can instantiate the models by providing the values to all parameters, so that the actual costs and benefits can be calculated.

ed. We note that since the values of the parameters are not necessarily single values, but ranges as well, the model is able to visualize the cost-benefit models, as functions of these values. In the tool, the decision-maker is provided with a list of all the parameters that affect a decision, and is asked to provide the expected values for the decision (see Figure 7.4.e). For instance, regarding the first parameter (i.e., “*Customer Support Hourly Rate*”) of the first benefit model (i.e., “*Decreased Need for Customer Support*”) the value has been set to 20\$, as the company suggests that the monthly rate of the average personnel on this position is 2,900\$ and the average man-month consists of 145 hours. We note that parameters (e.g., “*Testers Hourly Rate*”) that are involved in more than one models (e.g., “*Cost for Performing Regression Testing*” and “*Cost for Performing Acceptance and Integration Testing*”) need to be provided only once, and parameters that are among the default settings of the company (or project), e.g., lines of code, or amount of debt are automatically filled in by the tool.

Provide the values for the models' parameters

Decreased Need for Customer Support		Cost to re-write code	
Customer Support Hourly Rate	20	Cost per LoC	7.5
Hours saved while performing Customers Support	50	Lines of Code to be transferred	200

Increased Market Share		TD Extra Interest	
Expected Increase	0.2	CC Increase	0.5
Market Share	0.002	LoC Increase	1.6
Market Size	2000	LoC updated evry year	3000
Price	850.12	Percentage classes updated every year	0.8

Upgrade of Existing Licenses		Training	
Customers that Upgraded	40	Instructor Hourly Rate	50
Upgrade Cost	50	Training Sessions in Hours	8

Analyze!

Figure 7.4.e: Instantiate Model

Assessing the Decision. This final step enables the assessment of the decision (choose to either apply it or not), based on the *financial profit* of making the decision (total benefit – total cost). Upon the instantiation of the models, the decision maker is provided with the outcome of the cost-benefit analysis (see Figure 7.4.f).

Figure 7.4.f: Cost Benefit Analysis

In the top part of the UI (Figure 7.4.f), the decision-maker can see an overview of the assessed cost and benefits for every model, given the input provided in Figure 7. Regarding the first benefit model of Figure 7 (i.e., “*Decreased Need for Customer Support*”), in the top-right part of the UI (see Figure 7.4.f), we can see that the benefit is 1,000\$ (20\$ per hour: first parameter of the model; and 50 hours saved: second parameter—see Figure 7.4.e). The lower part of Figure 7.4.f appears only if the decision maker selects to tune the decision: If the assessment of the decision is not clear, the decision maker is provided with the option to select specific parameters and provide minimum and maximum values for them. For example, in our running case, the decision maker can observe that the decision to move the computations to GPU is largely non-profitable. We note that in the provided charts, the “*profit*” area is the one in which the “*red*” line is higher than the “*blue*” line (marked in the line chart). Therefore, they can check if there is a parameter, which if modified can constitute the decision profitable. In Figure 7.4.f, through the “*Select Parameter to Tune the Model*” feature, the user experiments on the cost of Training—based on the default cost model, the parameter of interest is “*Instructor Hourly Rate*”. Through the line charts, the user understands that they should hire an instructor for training, paid at most 32\$ per hour (point at which the blue line intersects with the red line), so that the decision is

beneficial¹⁹. We note that the tuning of the parameters is not considered as a decision itself (i.e., the user does not decide to hire a lower rate instructor), but only as a way to explore how to make a decision beneficial.

7.5 Empirical Methodology

Case study is an observational empirical method that is used for retrieving empirical evidence in a real-life context (Yin, 2003). To evaluate the proposed approach and tool to identify strengths and limitations, we conducted a case study. In this chapter we present the case study design, reported according to the template by Runeson et al. (2012).

7.5.1 Research Objectives & Research Questions

The goal of this study in terms of the Goal Question Metric (GQM) approach (van Solingen et al., 2002) is formulated as follows: “*analyze* the proposed decision-making approach and the corresponding tool *for the purpose of* evaluation *with respect to* their usefulness and usability *from the point of view of* software architects and managers, *in the context of* architectural decision-making”. Based on the abovementioned goal, we have derived three Research Questions (RQs):

RQ1: What is the current status of architectural decision-making approaches and tools within the case study organization?

Before performing the evaluation of the proposed approach and tool, we first need to understand the current way in which decisions are made. By evaluating the state of practice in architectural decision-making in the particular organization that we analyzed, we can understand if the proposed approach and tool treat existing limitations and retain the strong points. Thus, we investigate the involved stakeholders, the tools that are used, and the processes followed by the company. The benefits and drawbacks of using the already existing decision-making approaches are also discussed, to understand what already works well and what needs to be improved.

RQ2: How is the proposed approach evaluated by practitioners in terms of usefulness and ease-of-usage?

In this research question we explore the main aspects of the proposed approach, based on the specific context of the industrial partner. First, we explore the acceptance of the existing cost and benefit models that are introduced as default models in the approach. At the same time, we seek input for additional models that practitioners think that are missing from the default ones, aiming at evaluating the completeness of the provided default mod-

¹⁹ We note that the graphical representations of Figure 8 include only linear models as an example; the tool can handle any other function

els. Next, we proceed with the evaluation of the approach, based on the two basic aspects of the Technology Acceptance Model (TAM): perceived usefulness and perceived ease-of-usage.

RQ₃: *Does the developed decision-making tool meet the expectations of the practitioners?*

Having received input for the approach itself, in **RQ₃**, we focus on the offered tool support. Specifically, we investigate the usability (perceived ease-of-usage) of the tool and the extent to which it meets its functional requirements (perceived usefulness). Answering this question acts as a validation process for the tool and extends the body of knowledge on architectural decision-making. We note that, to answer **RQ₂**, we first need to demonstrate the approach to the subjects, so as to get a better understanding of its applicability and capabilities. Furthermore, to answer **RQ₃**, the subjects need use the tool; we do that by assigning two tasks to them and asking them to complete the tasks in a given time frame. Thus, we clearly separate the evaluation of the approach (**RQ₂**) from the tool itself (**RQ₃**).

7.5.2 Case and Subjects Selection

To answer the aforementioned questions, we performed an embedded multiple-case study in the software industry. The context of the case study is a Swedish SME, namely CNET, which is a software house specializing in research and innovation for Internet of Things technologies. The cases are two projects concerning the application of an IoT Box on two types of devices: PLCs in smart agriculture and bridges. IoT Box sensors gather vibrations, tensions, temperature etc., and send them to a cloud solution. The data is reported using the OGC (Open Geospatial Consortium) format and an open API allows smart phone apps to be created to render the data and process it for advanced analytics.

Participant	Role	Experience (in years)		
		Total	IoT Box	Decision-making
PM1	Project Manager	31	3	21
PM2	Project Manager	30	2	21
A1	Architect	24	2	18
A2	Architect	26	3	26

Figure 7.5.2: Study Demographics

The case study is embedded, in the sense that inside each case (IoT Box for smart agriculture and bridges), more than one unit of analysis have been studied. Specifically, the units of analysis correspond to the four participants of the case study: one project manager and one software/system architect per case. Some demographics of the participants are present-

ed in Table 7.5.2 (the experience is measured in years).

7.5.3 Data Collection

We collected qualitative data through different data collection methods, which are presented in Table 7.5.3.a and are further discussed below. For all research questions, method triangulation has been applied to increase the validity of the findings. Method triangulation refers to the technique of mixing more than one method to gather data (such as interviews, observations, questionnaires, and document inspection, etc.) to answer a research question, so as to reduce bias, and raise confidence in the results.

Collection Method	RQ ₁	RQ ₂	RQ ₃
Interview	X		X
Focus Group	X	X	
Questionnaire		X	
Observations / Task Analysis			X

Figure 7.5.3.a: Data Collection Methods per Research Question

The case study was executed as a one-and-a-half-day workshop that was held in the premises of CNET, and was conducted by the second and the fourth co-authors of the article. The detailed schedule and organization of the workshop is presented in Appendix C.1—Section 1. The workshop started with a brief introduction of the goal of the workshop and the scope of the method, and an introduction of the problem.

The goal of **RQ₁** is to understand the state-of-practice on architectural decision-making in the selected company. To achieve this goal, we first conducted interviews with the four participants. To guide the interview, we had a pre-planned guide (see Appendix C.1—Section 2). The interviews aimed at giving us some insight on the drawbacks of the current state-of-practice, so that we can check if these drawbacks are (at least partially) addressed by the proposed solution. Similarly, strengths of the current practices were identified and were used to check whether the developed approach would maintain such strengths. Next,

Figure 7.5.3.a Workshop Demonstration Scenario

we have performed a focus group, structured according to the guidelines provided by Kontio et al. (2007). During the **planning** of the focus groups, we defined the goals: i.e., understand current practices in decision-making. Regarding the **design**, the focus group lasted for 45' and the participants were the same as in the interviews. While **conducting** the focus group the discussion was focused on the quality attributes that are central to architectural decision-making, by the involved technical and managerial parameters, the way to aggregate them, and the financial aspects of the decision-making—see Section 3 in Appendix C.1.

Regarding **RQ₂** (evaluating the ADMIT method), we first demonstrate the application of the proposed cost-benefit analysis on a hypothetical decision to be applied in the IoT Box project of CNET. In particular, we illustrate the steps that would be needed to evaluate a decision to replace some CPU calculations with GPU calculations. This decision (as presented in Figure 7.5.3.a), would provide energy and performance benefits, but would cause the degradation of maintainability since the code would become more complex. We note that the demonstration scenario was pre-determined and served only as an illustration of the process. To build a cost-benefit model, we have developed three indicative cost and three benefit models, as presented in Appendix C.1—Section 4, through a presentation and a detailed analysis of costs and benefits.

<<Cost/Benefit Model Name>>

To what extent do you think that this model is accurate? (Strongly Disagree → Strongly Agree) – 5 scales

With what frequency do you think that this model will be used? (Very Rarely → Very Frequently) – 5 scales

How easy is to assess the parameters needed to instantiate this model? (Very Difficult → Very Easy) – 5 scales

Figure 7.5.3.b Questionnaire for RQ₂ (Structure)

After providing the demonstration on how the aforementioned models are instantiated and how the cost-benefit analysis problem can be solved, there are two data collection sessions. The first one is a questionnaire, in which the default cost-benefit models of the approach (see Chapter 7.3.2) are being evaluated, using the structure presented in Figure 7.5.3.b. All questions are available in Appendix C.1—see Section 5. For example, consider the model of “*Exploit New Market Opportunities*”. The monetization equation of this benefit (NEW_MARKET_SIZE X NEW_MARKET_SHARE X PRICE) is considered as accurate (given the accuracy of the values of the parameters); however, the practitioners might consider

that a single decision (in isolation) is not adequate for opening new market opportunities (thus they have assigned a low frequency).

<<Cost/Benefit Model Name>>

What additional parameters would you need?

Would you use a different aggregation method?

Do you agree with the weighting?

Figure 7.5.3.c Focus Group for RQ2 – Part 1 (Structure)

Similarly to before, the focus group was designed based on the guidelines of Kontio (2007). In particular, during the **planning** of the focus group we defined the goals: (a) to discuss the details of the default cost/benefit models; (b) to explore the need for extending the set of default cost/benefit models; and (c) to evaluate the proposed approach. Regarding the **design**, the focus group lasted for 90' with the same participants. While **conducting** the first round of discussion (related to goal (a)), we used the structure presented in Figure 7.5.3.c—see Appendix C.1 (Section 6, Part 1).

Next, in Part 2 of the focus group, we discussed goal (b), using the following questions as topics: “*What additional models would you need?*”, “*Please define the model (parameters, aggregation method, weights)*”, “*When would this model be used for?*”, and “*What stakeholders would need to be involved to instantiate the model?*”—see Appendix C.1, Section 6, Part 2. Finally, regarding goal (c), the discussion was around the key expected improvements and liabilities of the method (e.g., in terms of usability, timeliness, accuracy, generalizability). The questions that have been used as drivers for the discussion in the last part of the focus group, providing an overview of its internal organization, are presented in Appendix C.1—see Section 6, Part 3.

Regarding **RQ3** (evaluating the supporting tool), we have organized three main sessions. In the first session, we used the task analysis method (Lethbridge et al., 2005) by observing the participants performing two tasks on the tool, by plugging their laptops in a projector, and “recording” their rationale using the think-aloud-protocol (TAPS) (Lethbridge et al., 2005). Each participant was provided with a usage scenario and two tasks, as described in Appendix C.1—Section 5. Next, each participant, was interviewed with the goal of evaluating the functional requirements and the usability of the tool. To guide the interview, we have a pre-planned guide, as presented in Appendix C.1—Section 6. Finally, to assess the usability in a quantitative and more objective way, we used the System Usability Scale (SUS), which is a state-of-the-art method in the user interface design field (Brooke, 1996). SUS is reliable tool for measuring usability. It consists of a 10-item questionnaire with five

response options for respondents; from Strongly agree to Strongly disagree. Originally created by Brooke (1996), it allows UI/UX experts to evaluate a wide variety of products and services, including hardware, software, mobile devices, websites and applications. The items of evaluation are presented in Table 7.5.3.b. The participant's scores for each question are converted to a number, added together and then multiplied by 2.5 to convert the original scores of 0-40 to 0-100. Though the scores are 0-100, these are not percentages and should be considered only in terms of their percentile ranking. Based on the literature, SUS scores above 68 are considered above average and anything below 68 is below average (Brooke, 1996).

Table 7.5.3.b System Usability Scale

I think that I would like to use this system frequently	I thought this system was too inconsistent
I found the system unnecessarily complex	I felt very confident using the system
I thought the system was easy to use	I found the system very cumbersome to use
I think I would need the support of a technical person to be able to use this system	I would imagine that most people would learn to use this system very quickly
I found the various functions in this system were well integrated	I needed to learn a lot of things before I could get going with this system

7.5.4 Data Analysis

The dataset was inspected using lexical analysis for quantitative assessment and constant comparison for qualitative assessment. First, we transcribed all interview and focus group audio files, and we compiled them into a data set, including also the notes kept during the observation sessions. Then a lexical analysis took place: in particular, we have counted word frequency, and then searched for synonyms and removed irrelevant words. Then we coded the data set, i.e., categorized all pieces of text that were relevant to a particular theme of interest, and we grouped together similar codes, creating higher-level categories. The categories were created during the analysis process by both the first and the second author, and were discussed and grouped together through an iterative process in daily meetings. To visualize the outcome of this process word-clouds have been developed.

7.6 Results

In this chapter, we report the findings of this case study, organized by research question. Therefore, in Chapter 7.6.1, we present an overview of the state-of-practice in the company, accompanied by its perceived benefits and limitations. In Chapter 7.6.2, we focus on the evaluation of the approach, whereas in Chapter 7.6.3 on the tool evaluation.

7.6.1 State-of-the-Practice in Decision-making in CNET (RQ1)

In this chapter, we discuss the current state-of-practice in CNET for architectural decision-making. First, we describe two (recent) decisions that have been described by the participants, so as to set the context for the discussion. Next, we proceed with a summary of the current process, and a discussion of its limitations and advantages.

Context: As part of the case study, the following two decisions were discussed: (a) PM1 and A2 referred to a decision that led to the development of a diagnostics' tool software (alternative: hardware update, or no action), which is able to self-check the IoT box, and could be extremely useful in cases of connectivity problems, especially for far remote bridges, which are not easily accessible (also due to weather conditions). In this case the quality properties that have been optimized were: reliability and robustness; and (b) PM2 and A1 referred to a decision related to caching and pre-processing data of edge devices, before sending them to the back-end (the cloud infrastructure). This decision was important for the scalability of the Smart Agriculture project, in which the option was either to increase the use of cloud resources (more costly for end-customers and not scalable), or rewrite a large piece of the code.

The structure and the level of evidence in each decision was different, for each stakeholder, probably due to their level of seniority and experience. The first case (namely, the Smart Bridge System—see Figure 7.6.1.a) was discussed with PM1 and A2. The need to take a decision arose when the system (installed in a Bridge far away from Stockholm) stopped sending recordings. The project manager and the responsible architect had to decide if they would treat the problem as a hardware, a software, or a random problem.

Figure 7.6.1.a: Smart Bridge Decision Panorama

The initial thought of PM1 was described as follows: *“Based on my experience on similar problems in the past (by considering the higher cost of HW and the high probability that such a problem occurs again), I opted to go for treating the problem as a software problem, and build a diagnostic software to do the job”*. The decision was not taken only given the current situation, but by considering also the avoidance of future costs: *“Given the frequency of such connectivity failures, we believe that we have saved about 200-300K euros, by investing 2 weeks in the development of the diagnostics tool”*. Discussing the same system, A2 pointed out that apart from cost saving other parameters have been considered, such as the environment (*“Most of them are placed far away from Stockholm and reaching them regularly for maintenance can become very costly. Thus, it is important to keep them robust”*), the opinion of customers and domain experts (*“to make the aforementioned decisions, we needed to talk a lot to customers, developers and experts in the development of IoT boxes, especially to get informed on the data consumption of such devices, and the reading capacity of the cloud services”*). In this decision the stakeholders have not used any tools, or made any concrete calculations.

Regarding the second case, the Smart Agriculture project (see Figure 7.6.1.b), the two stakeholders exhibited a higher level of collaboration, since they were of different types:

Figure 7.6.1.b: Smart Agriculture Decision Panorama

one more technical and the other more managerial. In the specific case of data caching, the decision started from A1, who noticed high peaks in the usage of cloud resources when adding farms in the IoT system.

Similarly to the previous decision, the architect has also considered future versions of the system: *“The numbers seemed to be even more alerting by time-framing the problem in the upcoming year. Given the expected growth of the customers and the new installations in PLCs, it seemed to me that we need to make a decision on if we should rewrite the software using caching libraries, or if we shall leave it for the future and increase the cost for end-customers”*. A1, as a technical stakeholder person, could estimate the time required to re-write the code, but he was not able to assess the business perspective of this decision, so he needed to ask input from PM2: *“This decision was critical at that point, because from farming lots of data come in summer months, and if you do not react early, the system might ‘crash’ physically or economically in the peak of its usage. Thus, based on the estimates of A1, I decided to explore the business case more closely.”* The collaboration between the stakeholders ended up in an excel file with the technical calculations (however, no specialized tool for decision-making has been employed), enhanced with the financial estimations that were made after talking to the customers. In the end the decision was to prioritize for enabling the scaling of the economy, rather than adding new functionalities;

thus, customers were informed that in the upcoming sprints there will be no delivery of new functionalities, so as to decrease the cost per installation. Along this decision, the stakeholders, driven by the need to collaborate, used excel for data exchange and simple calculations, but in general they also described their process as ad-hoc and conceptual rather than a structured approach, that could be generalized, reproduced and reused.

Analysis of Current Process: Based on both the interviews and the focus groups, the decision-makers of CNET suggested that the current-state-of-practice is an almost purely conceptual process, which is based on existing evidence and numbers that stem from various sources. These however are not systematically synthesized, but mostly based on their experience. A lexical analysis of the transcripts from interviews and the focus group have revealed the most common terms that the participants used, while describing the current state-of-practice and evaluating it. The results of the lexical analysis are presented in Table 7.6.1. From Table 7.6.1, it becomes evident that that they are already considering financial aspects of the decisions, such as the obtained *benefits* and primarily the *costs* required to apply the decision. Nevertheless, we need to note that the extensive use of these two terms might be subject to bias, due to the nature of the study. Additionally, the customers (“*what the customers need*”) are crucial in the decision-making process, as well as *time*. We note that *time* is used under two perspectives by the participants: (a) as the time horizon for the decision, and (b) as the time required to perform an action. Finally, the participants highlighted the need for basing their decisions on *data / facts*, i.e. “*what the decision-maker really knows*”, and not the personal opinion of the manager, and what “*they thinks on a given problem*”.

Table 7.6.1: Terms Describing the Current State-of-Practice and its Evaluation

Terms	#	Terms	#
cost	93	support	34
customer	85	model	31
time	67	problem	26
need	57	functionality	25
data / fact	53	performance	23
information	51	issues	22
discussion	43	people	21
benefits	39	tool	21
quality	38	scaling	19
process	34	parameters	18

In a retrospective evaluation of the conceptual process that is used for decision-making in the previous examples, the involved stakeholders identified specific benefits and limita-

tions. On the one hand, the main ***benefit of the existing approach*** is its *simplicity* and the fact that it *saves time*. All participants agreed that the used conceptual process *exploits the knowledge of the decision maker*. In particular, A2 claimed that having a knowledgeable manager poses a benefit, since they always decide based on what they really know: *“It is good to base your decision on what you really know and what you have experience on. This brings more confidence to the final decision”*. Concluding, the participants agreed that this approach is manager-centric and relies on how knowledgeable the decision maker is. Nevertheless, it was explicitly stated that before applying any decision, the decision was discussed among high-level managers, being presented and supported by the decision-maker. On the other hand, ***limitations*** have also been identified. In particular, participants stated that the current approach:

- lacks *structure and control*. According to the stakeholders, this lack of structure and control can lead to less thorough thinking of the problem space, and eventually end up with not identifying all possible alternatives, or evaluation criteria. These problems can lead to the approach not scaling well in larger problems.
- does not enable decision *reuse*. The reused parameters and rationale, are not documented in any way, and therefore any similar decision has to be assessed from scratch.
- is not easy to *involve various roles* and *support collaboration*. Using this conceptual process does not enable getting data from the most knowledgeable person in the company, as PM2 described: *“Inevitably we cannot know everything! At various points of decision-making you need information that you do not have. In such cases it is important to have stakeholders directly involved in the process rather than acting as a hub of collecting knowledge”*.

might *induce bias* from the perspective of the decision-maker. In particular, it was claimed that when presenting and supporting the decision to other managers, the decision maker is always positively biased in persuading others that they have selected the most fitting alternative.

7.6.2 Evaluation of the ADMIT Architectural Decision-Making Approach (RQ2)

In this chapter, we focus on the evaluation of the proposed approach for supporting architectural decision-making. We note that despite the fact that we have developed an approach and an accompanying tool, we decouple the evaluation of the two; that is why we did not present the tool to the participants of the case study, while evaluating the approach. The approach was evaluated under two perspectives: first, we evaluated the default cost and benefit models, which are central to the success of the approach; second, we evaluate the

approach itself, and the extent to which it alleviates the limitations of the current decision-making approach in the company.

Evaluation of Default Cost and Benefit Models: The evaluation of the default models took place in two steps: first through a questionnaire that aimed at evaluating the *accuracy*, the anticipated frequency of actual usage (*applicability*), and the *ease* of applying the cost and benefit models (see questionnaire structure in Figure 7.5.3.b). The results are presented in Table 7.6.2. To aggregate the scores from the four practitioners, we added the value that they assigned (1: strongly disagree – 5: strongly agree / 1: very rarely – 5: very frequently, and 1: very difficult – 5: very easy). Thus, the total scores can vary from 4 to 20. In Table 7.6.2, the following notations have been used, similarly to a heatmap:

- with green cell shading, we highlight cost/benefit models that score between 16-20, i.e., the majority of votes ranges from “*Strongly Agree*” to “*Very Strongly Agree*”, “*Frequently*” to “*Very Frequently*”, or “*Easy*” to “*Very Easy*”.
- with light green cell shading, we highlight cost/benefit models that score between 12-16, i.e., the majority of votes ranges from “*Neutral*” to “*Strongly Agree*”, “*Neutral*” to “*Frequently*”, or “*Neutral*” to “*Easy*”.
- with grey cell shading, we highlight cost/benefit models that score between 8-12, i.e., the majority of votes ranges from “*Disagree*” to “*Neutral*”, “*Rarely*” to “*Neutral*”, or “*Difficult*” to “*Neutral*”.
- with red cell shading, we highlight cost/benefit models that score between 4-8, i.e., the majority of votes is: “*Strongly Disagree*” to “*Disagree*”, “*Very Rarely*” to “*Rarely*”, or “*Very Difficult*” to “*Difficult*”.

Table 7.6.2: Default Cost-Benefit Models Evaluation

Cost Model	Accuracy	Applicability	Ease
Costs for Hiring Quality Personnel	18	12	13
Costs for Performing Meetings	17	13	16
Costs of Training	12	13	14
Increase Preventive Maintenance Effort	14	14	13
Cost of Producing Documentation	14	14	12
Increase Cost for Project Planning	17	16	17
Cost of Prototyping	13	15	15
Cost for Reusing Proprietary Software	13	15	18
Cost for Performing Code Inspection	17	13	17
Cost for Performing Design Reviews	17	14	18
Cost for Performing Unit Testing	16	16	12
Cost for Using Quality Assurance Tools	15	14	15
Cost for Performing Quality Audits	16	11	16

Cost Model	Accuracy	Applicability	Ease
Cost for Performing Acceptance and Integration Testing	17	16	14
Cost for Performing Metrics-Based Quality Assurance	14	11	11
Cost for Performing Regression Testing	14	11	13
Necessary Rework due to Failures (e.g., Debugging, Quality Improvement)	16	13	13
Declining Market Positioning	13	12	8
Additional Costs for Marketing	11	12	11
Lost Opportunity Costs	15	12	7
Benefit Model	Accuracy	Applicability	Ease
Increased Market Share	15	11	8
Upgrade of Existing Licenses	18	15	15
Decreased Need for Customer Support	18	17	16

From the aforementioned table, we can observe that out of 23 models, only 1 seems to underperform (grey or red shadings—below *Neutral*) in accuracy, 4 in terms of frequency of usage, and 5 in terms of ease of instantiation. On the contrary, 22 out of 23 models, score higher than *Neutral* in terms of accuracy, 19 out of 23 in terms of applicability, and 18 out of 23 in terms of ease of instantiation.

Figure 7.6.2: Frequency of Evaluations in terms of Accuracy, Applicability, and Ease of Instantiation

The aforementioned findings are visualized in Figure 7.6.2, through a stacked frequency bar, for each evaluation parameter (i.e., accuracy, applicability, and ease of use). The results suggest, that cost models are accurate, applicable, and easy to instantiate. Among the examined evaluation parameters, the one achieving the lowest score (although still quite high) is the ease of instantiation. A possible interpretation for this outcome is that identify-

ing the values for all parameters is an aspect that requires input from various stakeholders, and is quite subjective. For example, “*Additional Costs for Marketing*” requires specialized marketing knowledge, for which only one of the participants was aware of.

As a counter measure, to improve these values, during the focus group, we discussed these cost / benefit models, and identified ways to improve their quality. For example, “*Costs for Performing Meetings*” could be improved by entering different rates for all participants, i.e., naming a-priori the participants in meetings and retain their exact hourly rate from HR services. This would improve both the accuracy and the ease of instantiation. The same strategy can also be applied in “*Cost for Performing Metrics-Based Quality Assurance*”. Regarding the most ‘problematic’ models, we have been advised to update “*Declining Market Positioning*”, so as to take into account future sale, whereas in “*Additional Costs for Marketing*” to also consider indirect costs (such as investment material, marketing campaign material, etc.) and not only personnel cost. In terms of missing cost and benefit models that could be reusable in many projects, the participants highlighted the need for models that are related to cost for improving security, and costs for improving the user interface of a systems, such as usability testing, which they consider different from “traditional” testing.

Evaluation of the Proposed Approach: In general, the participants considered the financial view of the approach as beneficial. In particular, the participants suggested that the aggregation of various views under the same unit (i.e., currency) solves two main issues:

- ***improvement of communication within company and management:*** Participants consider communication as a “*constant problem*”, since “*everyone thinks that his job is the most difficult and important one*”. So, the use of such an approach for aggregating different views can provide a “*global understanding*”, a “*global point of view*”, and “*set a common ground for discussion*” before reaching a decision. Also, setting a common ground is expected to “*bring balance between the various stakeholders*”. Also, the participants agreed that “*money is a great unit, that everyone understands*”. Therefore, misunderstandings and underestimation on the importance of parameters are avoided. The use of the “*breaking point is a nice feature*”, since practitioners suggested that it is very interesting to “*play*” with the values of variables, so as to check multiple scenarios, without any cost.
- ***handling of various qualities and trade-offs.*** The participants believe that the approach can help in “*conscious and objective decision-making*” for trade-offs, that it enables upfront thinking and helps in “*avoiding bias*”. Using this approach can lead in understanding if “*it is worthy taking a decision*”. Also, the approach enables finding a “*balance between the risk and cost of failure*”, especially in cases when the opinion of

various stakeholders (representing the different qualities) needs to be considered. However, even when using such a structured approach, they make some “*best guesses for the values of the parameters*”. So, experienced people are still needed, since experience cannot be subsumed by a structured decision-making approach.

Additionally, the participants have not found the logic of the approach itself complex, despite the fact that identifying the values of some parameters might be difficult. On the contrary, they believe that “*problem complexity (which is large) is now distributed to many people (i.e., the various stakeholders)*”. PM2 suggested that after reaching a “*maturity state in using the method*”, you can reach a “*golden-level of detail, in which you achieve a perfect balance between bureaucracy and gains*”. However, before reaching this level, it is important to build a culture among stakeholders that management “*shall not be central*”, but it shall be “*distributed to knowledge holders*”. Nevertheless, building this culture will be much easier with such a financial approach, since “*the use of money is helping in this direction, since people consider them important*”. Finally, all participants agreed that applying the approach is very easy after modeling the problem.

Regarding the applicability of the approach in specific contexts and the required competencies to correctly apply the approach, the participants were also positive. By also considering the software development method, the participants believed that “*it perfectly fits agility*”, since feedback can be given in iterations, models can be continuously updated, and “*architectural decisions or refactorings can be incorporated into sprints*”. Furthermore, the participants highlighted that the distribution of decision-making responsibility, does not require an “*omniscient project manager or decision maker*”, since each person could contribute the “*piece of knowledge*” that they know best. Finally, the “*reuse of decision-making that the approach offers*” enables less experienced decision-makers to “*learn from the experience of others*”. Apart from the aforementioned positive feedback, multiple useful suggestions and substantial feedback for improving the tool have been provided; two examples are given below. First, the participants agreed that a default setting should be provided to all stakeholders before providing their input, so that one does not change the parameters of others (e.g., the size of the company, the seniority and rate of individual employees should already be available before usage, the setup of meetings shall be constant). Second, the participants agreed that risk and time are important factors in this kind of decision-making, and calculations of the various factors should consider them. We note that the notion of risk as part of architectural decision-making has been extensively discussed by Poort and van Vliet (2012).

7.6.3 Tool Evaluation (RQ3)

In Chapter 7.6.2, we evaluated the proposed approach, in isolation from the developed tool, so that the approach evaluation is not biased by possible problems of the tooling. In this section, we focus on the tool per se, aiming at identifying its current strengths, but also weaknesses that can drive its future developments. The tool evaluation session was divided into three main phases: first we observed the practitioners in performing two tasks in the tool, so as to get an indication of their ability to use the tool unassisted; second, we asked them to fill in the SUS questionnaire; and third, we conducted some interviews to get qualitative data on their impression and suggestions for future developments. Below, we report the findings based on the two goals of this research questions: usefulness (based on the interviews) and ease of use (based on interviews, task observation and the SUS questionnaire).

Usefulness: The results of the evaluation of the anticipated functional requirements are summarized in Table 7.6.3. Specifically, in Table 7.6.3, we present the most commonly occurring keywords in the discussion of each functional requirement, and some quotes that back the claim. It is evident from the results that the usability of the tool can be improved, but the feedback is generally positive and many advantages of using the tool have been highlighted.

Table 7.6.3: Evaluation of Functional Requirements

FR	Quotes
Equation / Model Creation	<p>“The use of the tool is straightforward”</p> <p>“The tool provides reuse in terms of equations and parameter, but this shall expand to values, and ranges of values”</p> <p>“Time is an important parameter for equations, we should have the ability to add it as such”</p>
Visualization	<p>“Quite easy to get the breaking point of the decision”</p> <p>“2D plots is a reasonable way to show the results, I would not expect to more complex visualization”</p> <p>“Ranges of values could be replaced with a scrollbar”</p>
Parameter Tuning	<p>“Observing the contribution of equations in the loss/profit is a very nice feature”</p> <p>“The tuning of parameters gives you the opportunity to experiment, even with fields outside your expertise”</p>
Customization	<p>“It would be nice if I could copy/paste models”</p> <p>“The fact that I can add a pessimistic and an optimistic view is a good way to customize the model instances”</p>

As a tentative list of improvements, the participants have provided us with multiple suggestions that are currently under development in the tool. The proposed usability improvements are summarized below:

- decisions should be *exportable* in some common format, like *.pdf*, or *.xls* files

- the user should have the ability to *save values* for specific *parameters* that should be visible to all decisions of the same project, or company
- *save multiple results* for the same decision, so that you can compare in a visual way
- *handle communication between stakeholders* from within the tool, e.g., sending emails or private messages
- there should be a *details section* in the equations and results panels, so that you can add information / assumptions for values or parameters.

Ease of Use: Based on the observation session, we have been able to validate that the participants could correctly perform the tasks under consideration. All participants were able to follow the instructions and translate the instructions into tasks that need to be applied in the tool. Also, the way of thinking of developers confirmed their understanding of the approach, since researchers only had to intervene 3 times to correct the rationale of the participants. Such interventions were initiated by the practitioners, who explicitly mentioned that they had difficulty understanding something, and needed the help of the researchers to proceed. Since the interventions were different for each participant, no direct conclusion on the ease of use could be drawn; therefore, further details are omitted from the manuscript. We note, that all participants were able to build the models, instantiate them, and tune them, so as to experiment with the parameters and identify the best setup for future tasks. In terms of SUS, the participants rated the usability of the tool from 70% to 82%, which is considered sufficient for a research prototype tooling. The main negative points that stood out from the SUS questionnaire was the existence of some bugs, and the (expected) lack of confidence of the evaluators when using the tool for the first time. With respect to usability evaluation, we explored the effectiveness, the efficiency, and user satisfaction received from the tool. Regarding effectiveness, all participants were confident that they used the tool appropriately, so as to correctly complete the task. In particular, the participants highlighted that they could very easily navigate from the description of the decision to the results, since the tool was simple to use. The participants, were especially satisfied with the tuning of parameters, since it provides more confidence in the decision-making. However, they noted that experienced users should have the opportunity to work only through the keyboard, without the need for the mouse, in order to be more effective. In terms of efficiency the stakeholders suggested that the task took reasonable time for completion, including the time to check the decision. However, the time required, would be increased if the models were not given (though, this is something unrelated to the tool, but mostly related to the approach). Finally, with respect to user satisfaction, the end-users seemed very satisfied from the tool, since they could imagine themselves using it in their day jobs: “*I can see how I could use the tool in my work, I can even now imagine some cost models for the storage decision. It would be*

easy and interesting to build some cost models for that with the tool". Another participant, added a different parameter in the daily usage of the tool: *"I could do that. However, only for projects that run for many years"*. In terms of complexity all participants were satisfied with the simplicity of the tool; one of the participants claimed that *"the problem is certainly complex, but the tool is very simple"*. Therefore, they manifested that they would be able to use the tool without any technical support, or documentation: *"a try-and-error process seems to work well, because the tool is very simple and the functionalities are intuitive. If it also becomes robust, there will be no need for manual"*.

7.7 Discussion

In this chapter, we discuss the main findings of the study from three perspectives: (a) we compare the findings to existing literature; (b) we provide tentative interpretations of the main results; and (c) we provide useful implications for researcher and practitioners.

Comparison to Related Work: In this study we explore the process of decision-making related to architectural decision-making, by: (a) exploring current practices in industrial software development—based on a case study in a single company, (b) propose an approach for architectural decision-making, accompanied by tool support; and (c) evaluate the proposed approach and tool in an industrial setting. Regarding industrial practices used for architectural decision support, our results from interviews and focus groups with industrial practitioners are aligned with related work. First, we confirmed the findings of Mantuffel et al. (2016) that the majority of participants do not use a particular, systematic approach for decision-making. For the case of our study, the participants described the used approach as a conceptual process that is neither structured, nor documented. Second, the results of the study validated the claim of Poort and van Vliet (2012) that supporting decision-making with financial data (e.g., cost or risk) can facilitate the communication with business stakeholders. Regarding the proposed approach, our work complies with this of Seaman et al. (2012), who proposed to use the cost-benefit analysis method from economics, and integrate in the decision-making process the business perspective, as supported by de Almeida et al. (2018). Both aspects have been appreciated from the industrial stakeholders: on the one hand they suggested that using currency as a unit for presenting the results, is important to attract the attention of involved stakeholders, especially of managers; on the other hand, they suggested that customers and therefore business parameters are key aspects in decision-making. Finally, the proposed approach goes beyond related work, in the sense that: (a) it provides more detailed guidance on how the cost-benefit modeling will be performed, since it provides a set of reusable and highly-applicable (as suggested by our results) set of default models; (b) it provides a tool that guides the process of parameter selection, weighing, and the comparison of alternatives, as well the real-time tuning of the

models; and (c) aggregates all outputs under a unified unit (i.e., a currency) that is well-accepted by industrial stakeholders.

Comparison of the Proposed Approach to the Existing Approach: When answering *RQ₁*, we listed both the strengths and the drawbacks of the existing approach in the case company. Our plan was to assess whether the proposed approach is able to maintain these strengths and also address the drawbacks, at least partially. This assessment is summarized in Table 7.7, organized based on the pros and cons of the existing approach.

Table 7.7: Comparison of Proposed Approach to Existing Approach

	Existing Approach	Proposed Approach
Pros	Saves Time	The participants confirmed that the proposed approach and the tool were easy to understand, and the time required to apply the proposed approach is minimal, especially after the cost/benefit models have been designed. Nevertheless, all participants confirmed that the development of models is expected to be a time-consuming process, especially for the first decisions. It is expected that the time to develop models can be reduced by: (a) reusing decisions; (b) reusing parameters; and (c) further familiarization with the approach.
	Simplicity	The participants mentioned that despite the fact that the approached problem is a very complex one, the proposed approach is very simple and does not require any special training or manual.
	Exploits knowledge of decision-maker	The participants acknowledged that the decision-maker, no matter how knowledgeable he is, may not know every detail. So, it is more beneficial to combine their knowledge with data from the most accurate stakeholder.
Cons	Lack of structure and control	The participants suggested that the proposed approach provides a frame for setting up the mental process that they used, so as to be more structured and disciplined.
	Not enabling reusability	The participants claimed that the documentation provided by the approach and the imposed organization, constitute the decisions more reusable for future cases, either for revisiting them, or reusing “as-is”.
	Not supporting collaboration	The participants highlighted that there is always a need to get input from various stakeholders. The collaboration opportunities that the approach and the tool provide, enable sharing the responsibility of making the decision with various stakeholders. This reduces the complexity of the problem.
	Inducing bias	The fact that the decision-making is now a collective responsibility, relieves the decision maker from the burden to stand up for their decision in company meetings. This, makes everyone more open to change opinions and support different suggestions.

Implications to Researchers and Practitioners: The findings of the study have enabled us to provide various implications for researchers and practitioners. On the one hand, ***decision-makers*** (i.e., project managers, architects, etc.) are advised to use tools, more structured processes, and rely on data and facts while attempting to reach their decisions. A structured way to achieve this goal is the use of the proposed approach and tool, which frames the decision space and has received positive feedback from industrial stakeholders, when assessing architectural decisions. In addition to this, as a best practice, while making decisions, the experts have highlighted the involvement of various stakeholders (including customers), the monetization of decision parameters, and the consideration of time and risk as important parameters in the decision-making. On the other hand, there are various interesting future work opportunities for ***researchers***. First, researchers are strongly encouraged to combine both technical and business perspectives while proposing approaches for architectural decision-making, since placing more emphasis on any of them, neglects important factors that need to be weighed. Second, based on the findings of the study, we highlight as an interesting extension of this study the following: (a) replicate the study in a larger context; (b) perform more research on benefit models, which until now seem to be neglected compared to cost models; (c) investigate appraisal and failure costs in more detail, since they seem to lag, compared to prevention cost models; and (d) explore opportunities for supporting the ease of instantiating these models by using automated techniques, based on data stored in company databases (both technical and business ones). Furthermore, we believe that an interesting future work direction to this study would be to extend ADMIT so as to handle interdependencies among decisions. Such an extension would need to rely on a network of decisions, whose outcomes could: (a) be used as inputs to other decision; (b) constrain other decisions—e.g., one alternative could be rejected, based on a previous decision; or (c) impose a decision—i.e., one alternative must be selected, based on a previous decision (Kruchten et al., 2006). Finally, we would encourage researchers to investigate into more detail how time and risk (important parameters in decision-making, as claimed by the practitioners) could be used in such models: i.e., how they could modify and mix with models that are able to express all decisions in monetary terms. For instance, regarding time (since risk as a factor is already discussed in the literature (Poort and van Vliet, 2012), including time as a parameter for the “*Increased Market Share*” model, could entail adding a new parameter “*year of expected sales*”, which would stand for the timeframe in which the new market share would be applicable. This would not calculate the increase of revenue for one year, but for the lifetime of the product.

7.8 Threats to Validity

Potential threats to validity of the conducted study concern construct, external, and internal validity and reliability threats. Since the goal of the study is not to establish any causal relationship, but only to provide an initial exploration, we believe that internal validity is not a main concern for this study's validity.

Construct validity reflects to what extent the phenomenon under study really represents what is investigated according to the research questions (Runeson et al., 2012). To mitigate construct validity threats, we established a research protocol to guide the case study, which was thoroughly reviewed by two experienced researchers in the domain of empirical studies. Additionally, during the data collection process we aimed at data and method triangulation to avoid a wrong interpretation of a single data source. Another threat is the fact that the tool and the approach have been evaluated separately, and without a long-term usage of the tool before the study; this has introduced both negative or positive bias. On the one hand (*negative bias*), the evaluation of the stakeholders was probably stricter, since the users were completely inexperienced with the tool, they have probably faced more usability issues, compared to an evaluation that would have been performed after some training or self-training period. Therefore, we believe that the presented results, correspond to the worst case scenario of usage and evaluation. On the contrary (*positive bias*), the evaluation might have been positively biased by using only a demo session, in which unclear parts were explained by the researchers, making them easier to understand by the practitioners. Nevertheless, since practitioners had time to experiment alone with the tool/approach in **RQ₃**, we believe that possible problems while instantiating the cost/benefit models (i.e., providing values to the parameters) would have been identified and reported.

In terms of **external validity** (i.e., the generalizability of the findings derived from the sample (Runeson et al., 2012)), it is difficult to claim that the same results would be derived in other companies. However, emphasizing on analytical generalization we can report on mitigation actions, which allow us to argue that the findings are representative for other cases with common characteristics (especially for **RQ₂** and **RQ₃**). Specifically, the participants of the study were professional software engineers with various years of experience in software development. Regarding **RQ₁**, however, the results might be difficult to generalize outside CNET, in the sense that decision-making and chain of responsibilities in SMEs may be very different from one company to another. Based on this, **RQ₁** has been stated explicitly for the CNET context. Another threat to generalization is related to the evaluation of default cost/benefit models. In particular, we cannot claim that the evaluation results can be expanded to decisions involving other kind of cost/benefit models, since the default set has been developed based only on one study (Karg et al., 2011). Nevertheless, we need to note

that the study of Karg et al. (2011) is a secondary study that provides an overview of the field of poor-quality cost modelling for several years, and therefore does not correspond to the personal view of some authors, but to the cumulative view of the completed domain. Finally, the general applicability of the approach is threatened by the fact that not all cost and benefits can be monetized; thus, ADMIT is applicable only to decisions that rely on cost and benefit models that can be assessed in currency forms.

The *reliability* of the case study concerns the trustworthiness of the collected data and the analysis performed, to ensure that same results can be reproduced (Runeson et al., 2012). We support the reliability of our study by creating a rigor case study protocol and interview guides, which were tested through pilots. To minimize potential reliability threats during the data collection process, we preferred to ask open-ended questions and we requested (Runeson et al., 2012) motivation for the provided answers. To assure the correct and unbiased data analysis, three researchers collaborated during the whole analysis phase. Finally, we have internally archived all collected data (both raw and coded), due to a non-disclosure agreement with our industrial partner. On the other hand, interview and focus group guidelines are openly available in Appendix C.1.

7.9 Conclusion

Architectural decision-making is a field that deserves investigation, in the sense that the number of decisions that need to be made is high in large software systems and can have substantial impact on software evolution. Software architects face various cases, in which they need to make a decision and effectively select the most fitting solution to an architectural problem. To make this process structured, controlled, and reusable, in this paper we propose a decision-making approach and an accompanying tool. The approach relies on studying both technical and business parameters of the decision and aggregates them by representing them in the form of cost and benefit models, that use currency as a unit for the calculations. Therefore, all parameters need to be monetized before “entering” the decision-making mechanism. The proposed approach has been evaluated in an industrial setting, through an embedded single-case study. The results of assessing the current practices in the industrial partner confirmed the lack of structured approaches and use of tools in the industry for making architectural decisions. The participants have positively evaluated the proposed approach, highlighting the importance of mixing various types of input (coming from different stakeholders) and aggregating them, using a well-accepted unit, such as money. Along evaluation, many positive aspects of the approach and tool have been highlighted, but also various points of improvement have been mined. Based on the findings of the study, various useful implications to researchers and practitioners have been provided.

Chapter 8 – Conclusions and Future Work

This chapter discusses the main conclusions reached through this PhD project. In particular in Chapter 8.1, we answer the research questions, and highlight the main contributions. In Chapter 8.2, we consider future work.

8.1 Answers to Research Questions and Contributions

The problem statement that this thesis aims to address is stated in Chapter 1.3.1 as follows: “*Current TD management approaches are incomplete as they fail to take into account the financial aspects of the metaphor. This is particularly manifested through: (a) a lack of approaches and tools for monetizing TD Interest, following the actual definition of TD Interest; (b) a lack of establishing a relation between TD Interest and Principal; and (c) a lack of approaches that consider costs and financial benefits beyond TD Principal and Interest in the decision-making of TDM*”. To address this problem, we have asked five main research questions, which were subsequently answered in Chapters 2 to 7. Table 8.1.a provides an overview of these research questions, maps them to Chapters of this thesis, and highlights the achieved contributions beyond the state of the art. The response to each research question is further elaborated after Table 8.1.a.

Table 8.1.a: Thesis Contributions

Research Question	Chapter(s)	Contributions
RQ1: What are the financial aspects of TD in literature?	2	An overview of the state-of-research on TDM. The main novelty of the literature review is its focus on the finance view of TD. <i>Primary Outcome:</i> A glossary and visual illustration of main TD terms, from a financial perspective.
RQ2: How do software engineers perceive TD?	3	An overview of the state-of-practice on TDM. The main novelty of the state-of-practice (at the point of publication) is that the study was one of the first industrial surveys on how practitioners perceive TD. <i>Primary Outcome:</i> The core findings of this study are: (a) TDM is required for projects with a long-forecasted maintenance period; and (b) that source code and architecture TD are the most prominent for practitioners.

RQ3: How can TD Interest be Quantified?	4 and 5	<p>Provision of the first interest theory, accompanied by a specific instantiation at the source code level, relying on a financial theory tailored to map the technical nature of TD.</p> <p><i>Primary Outcome:</i> A framework and a tool for quantifying TD interest at the source code level.</p>
RQ4: What is the relation between TD Interest and TD Principal?	6	<p>Exploration of a tentative relation between TD Principal and TD Interest. We were the first to provide empirical evidence on the existence of an indirect interest rate.</p> <p><i>Primary Outcome:</i> We discovered that classes with similar levels of TD Principal tend to produce similar levels of TD Interest.</p>
RQ5: How can we consider costs and benefits in the decision making of TD?	7	<p>Provision of an approach that treats decision-making as a financial investment. The novelty of such an approach is its uniform treatment of technical and financial parameters and its use as a communication medium.</p> <p><i>Primary Outcome:</i> An architectural method and a tool that uses as input both technical and financial aspects in TD decision-making.</p>

RQ1: What are the financial aspects of TD in literature?

The main motivation for asking this research question was to understand the current state-of-the-art on Technical Debt Management research. In particular, we aimed at exploring the financial aspect (i.e., usage of terminology and approaches) of TD. To answer this research question, we have performed a systematic literature review. We found out that the research TD research community lacks a focus on the financial nature of TD: a very limited number of papers advance the state-of-the-art by taking the financial perspective into account. The most prominent shortcoming that has been identified is that there are only limited (and at an infancy level) approaches for managing TD Interest. The literature review also allowed us to: (a) propose a glossary of TD financial terms and (c) thoroughly discuss on how financial methods have been used in TDM.

RQ2: How do software engineers perceive TD?

Through this research question we aimed at understanding the circumstances under which TDM is considered as necessary in practice. To answer this research question, we have performed a survey with industry practitioners. The results suggest that: (a) TDM is important for long-lived projects for which maintenance is costly—this observation has led to the need of studying TD sustainability, i.e., for how long is the decision to take TD benefi-

cial; (b) TD at the source code and the architecture level are the most important TD types in practice; and (c) that TD cannot be treated in isolation, since there are various trade-offs among quality attributes, including maintainability (which in the QA most closely related to TD).

RQ₃: How can TD Interest be Quantified?

Given the lack of approaches that can be used to quantify (not proxy) TD Interest, we devoted a large part of this PhD project in this research endeavor. As input for providing a comprehensive approach for TD quantification, we relied on: (a) financial interest theories, (b) existing maintainability predictors, and (c) maintenance effort. As a first step, we have built a TD Interest theory, based on the Liquidity Interest Theory, and introduced the concept of the Breaking Point, i.e., the point in time at which TD Interest would accumulate so high, that the decision to undertake TD Principal, will no longer be beneficial. As a second step, we defined the basic constituent of TD Interest calculation: (a) maintainability; and (b) maintenance effort. The first parameter (maintainability) needs to be calculated not as an absolute number (as in the maintenance community), but as a relative number, compared to the maintainability of a system without TD Principal. This parameter corresponds to how more (relatively) difficult maintenance is, due to the existence of TD. The second parameter (maintenance effort) corresponds to the average maintenance effort that is usually spent in a given period. Multiplying the two parameters leads to the overhead in terms of effort that can be attributed to TD. The validity of both the TD Interest calculation, as well as the Breaking Point have been examined through an industrial case study, indicating a very strong correlation (approximately 0.8) with the practitioners' opinion.

RQ₄: What is the relation between TD Interest and TD Principal?

By exploiting the aforementioned approach for TD Interest calculation, we were able to assess the TD metaphor, through exploring if there is a relation between TD Principal and TD Interest. This relation is the cornerstone of debt in finance, in the sense that the two concepts are related linearly through the interest rate. Nevertheless, in TD interest rate has not yet been captured and its existence has not been validated. To explore if the notion of interest rate stands for TD, we have performed a case study on open source software projects, using the Mantel test analysis. The results of the analysis suggested that classes with similar levels of TD Principal, tend to produce similar levels of TD Interest. The relation between the two concepts has been validated as at least moderate (in most of the cases as strong) that is statistically significant. By analyzing the two constituents of TD Interest (introduced in RQ₂), we validated that the combination of maintenance effort and maintainability is more strongly related to TD Principal, compared to the two concepts in isolation.

Finally, by further focusing on the constituents of TD Principal and TD Interest, we were able to highlight that for TD repayment, emphasis should be placed on improving specific quality properties (i.e., coupling and size), which have proven to be linked to the concentration of more TD Principal.

RQ5: How can we consider costs and benefits in the decision making of TD?

Given the fact that TD-related decision-making must not only consider the technical view of TD (e.g., amount of TD Principal or TD Interest), but also the financial consequences of TD (e.g., increased market share), we have identified a need for a high-level decision-making approach that can uniformly treat financial and technical inputs. To make this process structured, controlled, and reusable, we proposed an architecture decision-making approach and an accompanying tool. The approach relies on studying both technical and business parameters of the decision and aggregates them by representing them in the form of cost and benefit models, that use currency as a unit for the calculations. Therefore, all parameters need to be monetized before “entering” the decision-making mechanism. The proposed approach has been evaluated in an industrial setting, through a case study. The results of assessing the current practices in an industrial partner confirmed the lack of structured approaches and use of tools in the industry for making architectural decisions. The participants have positively evaluated the proposed approach, highlighting the importance of mixing various types of input (coming from different stakeholders) and aggregating them, using a well-accepted unit, such as money.

8.2 Ongoing and Future Work

As future work opportunities that have emerged from this PhD thesis, we have identified three main directions, one from each sub-problem (i.e., RQ3, 4 and 5): (a) extend FITTED calculation to artifacts other than the source code, (b) explore the tentative existence of a causal relation between TD Principal and TD Interest, and (c) extend ADMIT to become a full-fledged decision-making approach.

8.2.1 Improvement of the FITTED framework

Although the proposed calculation of TD Interest has been industrially validated, there is still need for improvement:

- ***Applicability to other Artifacts:*** By considering that the current instantiation of FITTED is limited to code TD, we believe that an interesting extension would be towards quantifying interest for other types of TD, such as architecture, requirements or documentation TD. For instance, requirements debt could concern costs such as the delay of developing features, whereas architecture debt could involve the effort to change architecturally-

significant design decisions, or the effect of TD on other quality attributes or even on the social aspects of the organization. Nevertheless, given the level of abstraction of these concepts, we see this research work more qualitative (e.g., involving experts) than quantitative.

- **Existence of Past Breaking Points:** As the current calculation of Breaking Point stands (principal / interest), we calculate only future breaking points. This assessment is important for decision-making in terms of how much time is left to the system for refactoring, but it does not consider cases in which the breaking point has already been reached (once or more times). This is something that is already studied in economics, e.g., taking a loan with a large interest rate, can lead to a cumulative interest that sums to 3-4 times the amount of principal: for such loans, more than one breaking point exists.

8.2.2 Further Studying the Existence of Interest Rate

The establishment of a correlation between TD Interest and TD Principal and their various aspects, begs the question whether causality also exists between them. This is especially interesting for 3rd level aspects of TD Interest and 2nd level aspect of TD Principal: “*Does complexity, for example, cause understandability rules violations?*”; “*Does low cohesion and high coupling cause poorly written code?*”. Such a causal study could be designed and performed through a controlled experiment, in which the researchers would control the amount of TD Principal in several variations of a system, and seek the actual maintenance time (TD Interest) for different amounts and types of TD Principal. The necessity (and difficulty) of targeting causation, instead of correlation has been discussed in detail in Dagstuhl 2014 on “Software Development Analytics” (Di Penta, 2016, Menzies, 2016).

8.2.3 Improvement of the ADMIT approach

Regarding the proposed architectural decision-making approach, we believe that it would be interesting to explore opportunities for supporting the ease of instantiating these models by using automated techniques, based on data stored in company databases (both technical and business data). Furthermore, we believe that an interesting future work direction to this study would be to extend ADMIT so as to handle inter-dependencies among decisions. Such an extension would need to rely on a network of decisions, whose outcomes could: (a) be used as inputs to other decisions; (b) constrain other decisions—e.g., one alternative could be rejected, based on a previous decision; or (c) impose a decision—i.e., one alternative must be selected, based on a previous decision. Finally, we would like to investigate in more detail how time and risk (important parameters in decision-making, as claimed by the practitioners) could be used in such models: i.e., how they could mix with models that are able to express all decisions in monetary terms. For instance, including time as a parameter

for the “Increased Market Share” model, could entail adding a new parameter “years of expected sales”, which would stand for the timeframe in which the new market share would be applicable. This would not calculate the increase of revenue for one year, but for the lifetime of the product.

Appendix A – Chapter 2

A.1 Papers included in the review

- [P1] E. Allman, “Managing technical debt”, *Communication*, ACM, 55 (5), pp. 50-55, May 2012.
- [P2] E. Alzaghouli and R. Bahsoon, “CloudMTD: Using real options to manage technical debt in cloud-based service selection”, *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 55 - 62, 18 - 26 May 2013, San Francisco, USA.
- [P3] B. Bartonand and C. Sterling, “Manage Project Portfolios More Effectively by Including Software Debt in the Decision Process”, *Cutter IT Journal*, October 2010.
- [P4] E. Bavani, “Distributed Agile Testing and Technical Debt”, *Software*, IEEE Computer Society, 29 (6), pp. 28 - 33, November/December 2012.
- [P5] J. Bohnet and J. Dollner, “Monitoring code quality and development activity by software maps”, *2nd Workshop on Managing Technical Debt (MTD '11)*, ACM, pp. 9 - 16, Hawaii, USA, 21 - 28 May 2011.
- [P6] J. Brondum and L. Zhu, “Visualizing architectural dependencies”, *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 7 - 14, Zurich, Switzerland, 2 - 9 December 2012.
- [P7] N. Brown, Y.Cai, Y.Guo, R.Kazman, M. Kim, P. Kruchten, E. Lim, A. McCormack, R. Nord, I.Ozkaya, R.Sangwan, C. Seaman, K. Sullivan, and N.Zazworka, “Managing technical debt in software-reliant systems”, *Proceedings of the FSE/SDP workshop on Future of software engineering research*, ACM, pp. 47 – 52, New Mexico, USA, 7-8 November 2010.
- [P8] F. Buschmann, “To Pay or Not to Pay Technical Debt”, *Software*, IEEE Computer Society, 28 (6), pp. 29-31, November/December 2011.
- [P9] S. Chin, E. Huddleston, W. Bodwell, and I. Gat, “The Economics of Technical Debt”, *Cutter IT Journal*, October 2010.
- [P10] Z. Codabux and B. Williams, “Managing technical debt: An industrial case study”, *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 8-15, 18 - 26 May 2013, San Francisco, USA.
- [P11] P. Conroy, “Technical Debt: Where Are the Shareholders' Interests?”, *Software*, IEEE Computer Society, 29 (6), p. 88, November/December 2012.
- [P12] B. Curtis, J. Sappidi, and A. Szynekarski, “Estimating the Principal of an Application's Technical Debt”, *Software*, IEEE Computer Society 29 (6), pp. 34 - 42, November/December 2012.
- [P13] B. Curtis, J. Sappidi, and A. Szynekarski, “Estimating the size cost and types of Technical Debt”, *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 49 - 53, Zurich, Switzerland, 2 - 9 December 2012.
- [P14] J. Davis and T. Andersen, “Surviving the Economic Downturn”, *Agile Conference 2009 (AGILE '09)*, IEEE Computer Society, pp. 245 - 250, Chicago, USA, 24 - 28 August 2009.
- [P15] R. J. Eisenberg, “A threshold based approach to technical debt”, *ACM SIGSOFT Software Engineering Notes*, ACM, 37 (2), pp. 1 - 6, March 2012.

- [P16] N. Ernst, "On the role of requirements in understanding and managing technical debt", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 61 - 64, Zurich, Switzerland, 2 - 9 December 2012.
- [P17] D. Falessi, M. Shaw, F. Shull, K. Mullen, and M. Keymind, "Practical considerations challenges and requirements of tool-support for managing technical debt", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 16 - 19, San Francisco, USA, 18 - 26 May 2013.
- [P18] I. Gat and J. D.Heintz, "From assessment to reduction: how cutter consortium helps rein in millions of dollars in technical debt", *2nd International Workshop on Managing Technical Debt (MTD '11)*, ACM, pp. 24 - 26, Hawaii, USA, 21 - 28 May 2011.
- [P19] J.M. Golden, "Transformation Patterns for Curing the Human Causes of Technical Debt", *Cutter IT Journal*, October 2010.
- [P20] R. Gomes, C. Siebra, G. Tonin, A. Cavalcanti, F.Q. da Silva, A.L. Santos, and R. Marques, "An extraction method to collect data on defects and effort evolution in a constantly modified system", *2nd International Workshop on Managing Technical Debt*, ACM, pp. 27 - 30, Hawaii, USA, 21 - 28 May 2011, ACM, New York, NY, USA, 2011.
- [P21] D. R. Greening, "Release Duration and Enterprise Agility", *46th Hawaii International Conference on System Sciences (HICSS-46)*, IEEE Computer Society, pp. 4835 - 4841, Hawaii, USA, 7 - 10 January 2013.
- [P22] J. de Groot, A. Nugroho, T. Back, and J. Visser, "What is the value of your software?", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 37 - 44, Zurich, Switzerland, 2 - 9 December 2012.
- [P23] Y. Guo and C. Seaman, "A portfolio approach to technical debt management", *2nd International Workshop on Managing Technical Debt*, ACM, pp. 31 - 34, Hawaii, USA, 21 - 28 May 2011.
- [P24] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F. da Silva, A.L. Santos, and C. Siebra, "Tracking technical debt - An exploratory case study", *27th International Conference on Software Maintenance (ICSM '11)*, IEEE Computer Society, pp. 528 - 531, Williamsburg, Virginia, USA, 25 September - 1 October 2011.
- [P25] J. D. Heintz, "Modernizing the DeLorean System: Comparing Actual and Predicted Results of a Technical Debt Reduction Project", *Cutter IT Journal*, October 2010.
- [P26] J. Holvitie and V. Leppanen, "DebtFlag: Technical debt management with a development environment integrated tool", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 20 - 27, 18 - 26 May 2013, San Francisco, USA.
- [P27] C. Izurieta and J. M.Bieman, "A multiple case study of design pattern decay grime and rot in evolving software systems", *Software Quality Journal*, SpringerLink, 21 (2), pp. 289 - 323, June 2013.
- [P28] M. Kaiser and G. Royse, "Selling the Investment to Pay Down Technical Debt: The Code Christmas Tree", *Agile Conference 2011 (AGILE '11)*, IEEE Computer Society, pp. 175 — 180, Utah, USA, 8 - 12 August 2011.
- [P29] T. Klinger, P.Tarr, P. Wagstromand, and C. Williams, "An enterprise perspective on technical debt", *2nd International Workshop on Managing Technical Debt*, ACM, pp. 35 - 38, Hawaii, USA, 21 - 28 May 2011.

- [P30] S. Koolmanojwongand, and J.A. Lane, "Enablers and Inhibitors of Expediting Systems Engineering", *11th Annual Conference on Systems Engineering Research (CSER '13)*, Procedia Computer Science, Elsevier, 16, pp. 483 - 491, Atlanta, USA, 19 - 22 March 2013.
- [P31] V. Krishna and A. Basu, "Minimizing Technical Debt: Developer's viewpoint", *International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA '12)*, IEEE Computer Society, pp. 1 - 5, Chennai, India, 19 - 21 December 2012.
- [P32] P. Kruchten, R. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice", *Software*, IEEE Computer Society 29 (6), pp. 18 - 21, November/December 2012.
- [P33] O. Ktata and G. Lévesque, "Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?", *3rd C* Conference on Computer Science and Software Engineering (C3S2E '10)*, ACM, pp. 101 - 107, Montreal, Canada, 19 - 20 May 2010.
- [P34] J. Letouzey and M. Ilkiewicz, "Managing Technical Debt with the SQALE Method", *Software*, IEEE Computer Society 29 (6), pp. 44 — 51, November/December 2012.
- [P35] J. L. Letouzey, "The sqalet method for evaluating technical debt", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 31 - 36, Zurich, Switzerland, 2 - 9 December 2012.
- [P36] E. Lim, N. Taksande, and C. Seaman, "A Balancing Act: What Software Practitioners Have to Say about Technical Debt", *Software*, IEEE Computer Society 29 (6), pp. 22 - 27, November/December 2012.
- [P37] R. Marinescu, "Assessing technical debt by identifying design flaws in software systems", *Journal of Research and Development*, IBM, 56 (5), pp. 1 - 13, September/October 2012.
- [P38] J. D. McGregor, J. Monteith, and J. Zhang, "Technical debt aggregation in ecosystems", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 27 - 30, Zurich, Switzerland, 2 - 9 December 2012.
- [P39] J. Monteith and J. McGregor, "Exploring software supply chains from a technical debt perspective", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 32 - 38, 18 - 26 May 2013, San Francisco, USA.
- [P40] J. Morgenthaler, M. Gridnev, R. Sauciuc, and S. Bhansali, "Searching for build debt: Experiences managing technical debt at Google", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, pp. 1 - 6, Zurich, Switzerland, 2 - 9 December 2012.
- [P41] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In Search of a Metric for Managing Architectural Technical Debt", *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, IEEE Computer Society, pp. 91-100, Helsinki, Finland, 20 – 24 August 2012.
- [P42] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest", *2nd International Workshop on Managing Technical Debt (MTD' 11)*, ACM, pp. 1 - 8, Hawaii, USA, 21 - 28 May 2011.
- [P43] K. Power, "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", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 28 - 31, San Francisco, USA, 18 - 26 May 2013.

- [P44] K. Pugh, "The Risks of Acceptance Test Debt," *Cutter IT Journal*, October 2010.
- [P45] N. Ramasubbu and C. Kemerer, "Towards a model for optimizing technical debt in software products", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 51 - 54, San Francisco, USA, 18 - 26 May 2013.
- [P46] M. Reddy, "Chapter 4 - Design", Morgan Kaufmann, Boston, 2011, pp. 105 - 150.
- [P47] D. Rooney, "Technical Debt: Challenging the Metaphor", *Cutter IT Journal*, October 2010.
- [P48] Y. Rubin, S. Kallner, N. Guy, and G. Shachor, "Restraining Technical Debt when Developing Large-Scale Ajax Applications", *1st International Conference on Building and Exploring Web Based Environments (WEB' 13)*, IARIA XPS Press, pp. 13 - 18, Seville, Spain, 27 January – 1 February 2013.
- [P49] P. S. M. dos Santos, A. Varella, C. R. Dantas and D. B. Borges, "Visualizing and Managing Technical Debt in Agile Development: An Experience Report", *Lecture Notes in Business Information Processing*, Springer, 149, pp. 121 - 134, 2013
- [P50] K. Schmid, "A formal approach to technical debt decision making", *9th International Conference on Quality of Software Architectures (QoSA' 13)*, ACM, pp. 153 - 162, Vancouver, Canada, 17 – 21 June 2013.
- [P51] K. Schmid, "On the limits of the technical debt metaphor some guidance on going beyond", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 63 - 66, San Francisco, USA, 18 - 26 May 2013.
- [P52] C. Seaman and Y. Guo, "Measuring and monitoring technical debt", *Advances in Computers*, Elsevier, 82, pp. 25 - 46, 2011.
- [P53] C. Seaman, Y. Guo, N. Zazworka, F. Shull, C. Izurieta, Y. Cai and A. Vetró, "Using technical debt data in decision making: Potential decision approaches", *3rd International Workshop on Managing Technical Debt (MTD' 12)*, IEEE Computer Society, pp. 45 - 48, Zurich, Switzerland, 5 June 2012.
- [P54] A. C. Shafer, "Infrastructure Debt: Revisiting the Foundation", *Cutter IT Journal*, 2010.
- [P55] S. Shah, M. Torchiano, A. Vetró, and M. Morisio, "Exploratory testing as a source of testing technical debt", *IT Professional*, IEEE Computer Society, 16 (3), pp. 44 – 51, March 2013.
- [P56] T. Sharma, "Quantifying Quality of Software Design to Measure the Impact of Refactoring", *36th Annual Computer Software and Applications Conference Workshops (COMPSACW' 12)*, IEEE Computer Society, pp. 266 - 271, Izmir, Turkey, 16 – 20 July 2012.
- [P57] C. S. Siebra, G. S. Tonin, F. Q. Silva, R. G. Oliveira, A. L. Junior, R. C. Miranda, and A. L. Santos, "Managing technical debt in practice: an industrial report", *6th International Symposium on Empirical Software Engineering and Measurement (ESEM' 12)*, ACM, pp. 247 - 250, Lund, Sweden, 19 – 20 September 2012.
- [P58] W. Snipes, B. Robinson, Y. Guo and C. Seaman, "Defining the decision factors for managing defects: A technical debt perspective", *3rd International Workshop on Managing Technical Debt (MTD' 12)*, IEEE Computer Society, pp. 56 - 60, Zurich, Switzerland, 5 June 2012.
- [P59] R. Spinola, N. Zazworka, A. Vetró, C. Seaman, and F. Shull, "Investigating technical debt folklore: Shedding some light on technical debt opinion", *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 1 - 7, San Francisco, USA, 18 - 26 May 2013.

- [P60] M. G. Stochel, M. R. Wawrowski, and M. Rabiej, "Value-Based Technical Debt Model and Its Application", *7th International Conference on Software Engineering Advances (ICSEA' 12)*, XPerT Publishing Service, pp. 205 - 212, Lisbon, Portugal, 18 – 23 November 2012.
- [P61] T. Theodoropoulos, M. Hofbergand D. Kern, "Technical debt from the stakeholder perspective", *2nd International Workshop on Managing Technical Debt (MTD' 11)*, ACM, pp. 43 - 46, Hawaii, USA, 21 - 28 May 2011.
- [P62] A. Vetró, "Using automatic static analysis to identify technical debt", *34th International Conference on Software Engineering (ICSE' 14)*, IEEE Computer Society, pp. 1613 - 1615, Zurich, Switzerland, 2 – 9 June 2012.
- [P63] K. Wiklund, S. Eldh, D. Sundmark and K. Lundqvist, "Technical Debt in Test Automation", *5th International Conference on Software Testing, Verification and Validation (ICST' 12)*, IEEE Computer Society, pp. 887 - 892, Montreal, Canada, 17 – 21 April 2012.
- [P64] R. J. Wirfs-Brock, "Skills for the agile designer: seeing shaping and discussing design ideas", *25th International Conference Companion on Object Oriented Programming Systems Languages and Applications*, ACM, pp. 323 - 326, Nevada, USA, 17 – 21 October 2010.
- [P65] J. Xuan, Y. Hu and H. Jiang, "Debt-Prone Bugs: Technical Debt in Software Maintenance", *International Journal of Advancements in Computing Technology*, 4 (19), pp. 453 - 461, 2012.
- [P66] N. Zazworka, C. Seaman, and F. Shull, "Prioritizing design debt investment opportunities", *2nd International Workshop on Managing Technical Debt (MTD' 11)*, ACM, pp. 39 - 42, Hawaii, USA, 21 - 28 May 2011.
- [P67] N. Zazworka, M. Shaw, F. Shull, and C. Seaman, "Investigating the impact of design debt on software quality", *2nd International Workshop on Managing Technical Debt (MTD' 11)*, ACM, pp. 17 - 23, Hawaii, USA, 21 - 28 May 2011.
- [P68] N. Zazworka, R. O. Spanola, A. Vetró, F. Shull and C. Seaman, "A case study on effectively identifying technical debt", *17th International Conference on Evaluation and Assessment in Software Engineering*, ACM, pp. 42 - 47, Porto Galinhas, Brazil, 14 – 16 April 2013.
- [P69] N. Zazworka, A. Vetró, C. Izurieta, S. Wong, Y.Cai, C. Seaman and F. Shull, "Comparing four approaches for technical debt identification", *Software Quality Journal*, Springer, 22 (3), pp. 403 – 426, September 2014.

A.2 Collected Data for RQ2.2

Study	Brief Approach Description	Software Eng. Technologies	Financial Terms	Financial Approach	TDM Activity
[P2]	The study focuses on the TD created by web services substitution, based on Real Options. Using a two-step binomial tree approach, it quantifies TD and identifies the time period during which TD is cleared out.	-	Amount of debt Present Value Future Value	Real Options	Measurement
[P9]	The article calculates TD as a function of simple and compound interest, and the expected years of software's active development and maintenance. Simple interest can be determined	Static Analysis Source Code Testing Design Smells	Simple interest Compound Interest Amount of debt	Software Economics	Measurement

Study	Brief Approach Description	Software Eng. Technologies	Financial Terms	Financial Approach	TDM Activity
	by ongoing product maintenance costs, while compound interest rate may be measured by the increase in technical debt over time.				
[P12]	This article proposes the estimation of Technical Debt principal as a function of the number of should-fix violations in an application, the hours needed to fix each violation and the labor cost.	AIP Tool Static Analysis Architecture Source Code	Principal	Software Economics	Measurement
[P13]	This paper calculates the principal of TD with a formula that takes into consideration three variables—the number of must-fix problems in an application, the time required to fix each problem, and the cost for fixing a problem.	AIP Tool Static Analysis Architecture Source Code	Principal	Software Economics	Measurement
[P22]	TD equals Repair Effort that is the effort needed to improve the quality of a system to the ideal level. Additionally, it calculates Repair Effort as a function of 3 variables: an estimated percentage of LOC that needs to be changed in order to reach an ideal level of quality, the effort required to rebuild the system from scratch and a ratio discount for code volume that does not need to be rebuilt.	SIG/TÜV Model	Amount of debt Interest	Software Economics	Measurement
[P23]	The paper suggests the quantification of Technical Debt principal by the effort required to resolve Technical Debt Items (i.e. source code that needs to be modified, test cases that need to be exercised, documentation that needs to be updated, etc.). TD interest is related to interest standard deviation (based on Portfolio Management Theory), that is the probability of this interest to incur. All three parameters are estimated using historical data.	Source Code Testing Documentation	Principal Interest	Portfolio Management	Measurement
[P24]	The paper presents a case study where Technical Debt principal, interest amount and interest probability are estimated based on historical effort data collected from the project documentation.	COCOMO Static Analysis	Principal Interest	Software Economics	Measurement
[P41]	The paper assesses the amount of TD and interest through a model that computes the rework cost associated with each new architectural element implemented in every new release.	Architecture	Amount of debt Interest	Software Economics	Measurement
[P42]	This paper defines the amount of TD as the Rework Effort and calculates it as a function of	SIG/TÜV Model Static Analysis	Amount of debt Interest	Software Economics	Measurement

Study	Brief Approach Description	Software Eng. Technologies	Financial Terms	Financial Approach	TDM Activity
	Rework Function, Rebuild Effort and Refactoring Adjustment. TD interest is defined as the difference of Maintenance Effort between a particular quality and the ideal quality level.	Source Code	Net Present Value		
[P49]	The paper reports a case where technical debt is quantified in terms of effort needed to fix it, by the use of SonarQube tool.	SonarQube Tool	Amount of debt	Software Economics	Measurement
[P50]	The study provides formalization for the quantification of TD. TD is measured in terms of evolution costs and is defined as the difference between additional development cost introduced in an evolution and the respective cost required in an optimal implementation.	Architecture	Amount of debt	Software Economics	Measurement
[P57]	In a case study, the paper estimates debt, interest and debt payment as the difference in effort between two or more alternative decisions.	-	Amount of debt Interest Payment	Software Economics	Measurement
[P60]	The study proposes a three-layered model aligned with a typical software product development lifecycle (based on P-Diagram approach on risk assessment).	SonarQube Tool Wisdom of Crowds ATAM Architecture Source Code	Amount of debt ROI	Value-based	Measurement

A.3 Collected Data for RQ_{2.2}

Study	Brief Approach Description	Software Eng. Technologies	Financial Approach	TDM Activity
[P2]	The study uses a Binomial Option – Based Approach to manage Technical Debt. Binomial trees are built in order to indicate if and when an alternative can clear out Technical Debt and add value.	-	Real Option	Repayment
[P8]	The article proposes the consideration of Technical Debt's financial benefits and business advantages as well as of the costs of servicing or repaying the debt in order to manage it efficiently.	-	Cost/Benefit	Repayment
[P11]	The article suggests that TD should be handled as financial obligation and should be recorded on the firm's accounting books.	-	Accounting	Monitoring
[P23]	This study proposes TDM based on Portfolio Management and treats TD items as assets. For each item, it is needed to determine if it is better to keep it or to pay it off. A risk level is set and the model generates the optimal portfolio (A) of the TD items. The items not chosen for the new portfolio (A') are those that need to be paid off.	Source Code Testing Documentation	Portfolio Management	Repayment

Study	Brief Approach Description	Software Eng. Technologies	Financial Approach	TDM Activity
[P24]	This paper applies cost/benefit analysis on a case study in order to demonstrate how adequate Technical Debt management can save a firm from high cost decisions.	COCOMO Static Analysis Documentation	Cost/Benefit	Monitoring
[P32]	The article proposes Net Present Value theory as the most promising financial method to apply on Technical Debt management and decision making.	-	Value-based	Repayment
[P41]	The paper suggests that Technical Debt management should take into consideration both value and cost of the project and also include elements of Total Cost of Ownership Management in order to enable efficient decision making on repaying Technical Debt.	Architecture	Value-based	Repayment
[P42]	Using a case study, the paper defines the repair effort to improve the quality of a system as an investment. Thus, it proposes the assessment of the worthiness of such an investment by looking at its ROI or Net Present Value.	SIG/TÜV Model Static Analysis Source Code	ROI Net Present Value	Repayment
[P43]	This paper proposes the consideration of the development's team capacity as an investment portfolio that can be managed as a Real Options portfolio. Technical Debt reduction should be a significant investment in the team's portfolio.	Defect-Density Functionality Performance	Real Options	Monitoring
[P45]	The paper provides a decision framework to assess the overall benefit (or loss) of accumulating technical debt, based on platform-based product development and customer adoption curve.	Functionality	Marketing	Monitoring
[P50]	The study proposes a decision-making approach on paying off technical debt items, according to the refactoring cost, the relative technical debt per refactoring and per evolution step and, finally, the probability that the evolution step will be performed.	Architecture	Cost/Benefit	Repayment
[P53]	The paper suggests the management of Technical Debt by the use of Cost – Benefit analysis, Analytic Hierarchy Process, Portfolio approach and Options theory.	Refactoring Design Smells	Cost/Benefit Portfolio Management Real Options	Prioritization Prioritization Repayment
[P58]	The paper identifies different kinds of costs associated with handling defects and investigates the criteria according to which a defect is decided to be fixed. Based on these findings, it proposes a cost – benefit analysis for decision making on fixing the defects.	Defect-Density Testing	Cost/Benefit	Repayment

Study	Brief Approach Description	Software Eng. Technologies	Financial Approach	TDM Activity
[P60]	By using Architecture Tradeoff Analysis Method (ATAM) and portfolio management, the paper proposes a model of Decision-making at the architecture level. Profitability aspect needs to be taken into account when managing; organization aims at optimal ROI.	SonarQube Tool Wisdom of Crowds ATAM Architecture Source Code	Portfolio Management	Monitoring
[P66]	This paper implements a cost – benefit approach in order to identify technical debt items that should be refactored first, by ranking the value and interest of design debt caused by god classes.	Refactoring Design Smells Defect-Density	Cost/Benefit	Prioritization

A.4 Primary Studies Quality Assessment

Study	Question Score					Total	Study	Question Score					Total
	1	2	3	4	5			1	2	3	4	5	
[P1]	1.00	1.00	0.00	0.00	0.00	2.0	[P36]	1.00	1.00	0.80	0.00	0.00	2.8
[P2]	1.00	1.00	0.20	1.00	0.50	3.7	[P37]	1.00	1.00	0.80	0.00	0.50	3.3
[P3]	1.00	0.50	0.00	0.00	0.50	2.0	[P38]	1.00	1.00	0.00	0.00	0.00	2.0
[P4]	1.00	1.00	0.40	0.00	0.50	2.9	[P39]	1.00	1.00	0.60	0.00	0.50	3.1
[P5]	1.00	1.00	0.80	0.00	0.50	3.3	[P40]	1.00	1.00	1.00	0.00	0.50	3.5
[P6]	1.00	1.00	0.60	0.00	0.50	3.1	[P41]	1.00	1.00	0.20	1.00	1.00	4.2
[P7]	1.00	1.00	0.40	0.00	0.50	2.9	[P42]	1.00	1.00	0.80	1.00	1.00	4.8
[P8]	1.00	1.00	0.00	1.00	0.50	3.5	[P43]	1.00	1.00	0.20	1.00	1.00	4.2
[P9]	1.00	1.00	0.00	1.00	1.00	4.0	[P44]	1.00	1.00	0.00	0.00	0.50	2.5
[P10]	1.00	1.00	0.80	0.00	0.50	3.3	[P45]	1.00	1.00	0.00	1.00	1.00	4.0
[P11]	1.00	1.00	0.00	1.00	0.50	3.5	[P46]	1.00	1.00	0.60	0.00	0.50	3.1
[P12]	1.00	1.00	0.80	1.00	1.00	4.8	[P47]	1.00	1.00	0.00	0.00	0.00	2.0
[P13]	1.00	1.00	0.80	1.00	1.00	4.8	[P48]	1.00	1.00	0.60	0.00	0.50	3.1
[P14]	1.00	0.50	1.00	0.00	0.50	3	[P49]	1.00	1.00	0.80	1.00	1.00	4.8
[P15]	1.00	1.00	0.20	0.00	0.50	2.7	[P50]	1.00	1.00	0.20	1.00	1.00	4.2
[P16]	1.00	1.00	0.00	0.00	0.50	2.5	[P51]	1.00	1.00	0.00	0.00	0.00	2.0
[P17]	1.00	1.00	0.40	0.00	0.00	2.4	[P52]	1.00	1.00	0.20	0.00	0.50	2.7
[P18]	1.00	1.00	0.80	0.00	0.50	3.3	[P53]	1.00	1.00	0.00	1.00	1.00	4.0
[P19]	1.00	1.00	1.00	0.00	0.50	3.5	[P54]	1.00	1.00	0.00	0.00	0.50	2.5
[P20]	1.00	1.00	0.80	0.00	0.50	3.3	[P55]	1.00	1.00	0.00	0.00	0.50	2.5
[P21]	1.00	1.00	0.00	0.00	0.50	2.5	[P56]	1.00	0.50	0.60	0.00	0.50	2.6

Study	Question Score					Total	Study	Question Score					Total
	1	2	3	4	5			1	2	3	4	5	
[P22]	1.00	1.00	0.80	1.00	1.00	4.8	[P57]	1.00	1.00	1.00	1.00	0.50	4.5
[P23]	1.00	1.00	0.00	1.00	0.50	3.5	[P58]	1.00	1.00	0.80	1.00	1.00	4.8
[P24]	1.00	1.00	0.80	1.00	1.00	4.8	[P59]	1.00	1.00	0.60	0.00	0.00	2.6
[P25]	1.00	1.00	0.80	0.00	0.50	3.3	[P60]	1.00	1.00	0.20	1.00	1.00	4.2
[P26]	1.00	1.00	0.00	0.00	0.50	2.5	[P61]	1.00	1.00	0.00	0.00	0.50	2.5
[P27]	1.00	0.50	0.80	0.00	0.50	2.80	[P62]	1.00	1.00	0.60	0.00	0.50	3.1
[P28]	1.00	1.00	0.80	0.00	0.50	3.3	[P63]	1.00	1.00	0.80	0.00	0.50	3.3
[P29]	1.00	1.00	0.40	0.00	0.50	2.9	[P64]	1.00	0.50	0.40	0.00	0.50	2.4
[P30]	1.00	0.50	0.00	0.00	0.50	2.0	[P65]	1.00	1.00	0.60	0.00	0.50	3.1
[P31]	1.00	1.00	1.00	0.00	0.50	3.5	[P66]	1.00	1.00	0.20	0.00	0.50	2.7
[P32]	1.00	1.00	0.40	1.00	0.50	3.9	[P67]	1.00	1.00	0.80	0.00	0.00	2.8
[P33]	1.00	1.00	0.00	0.00	0.50	2.5	[P68]	1.00	1.00	0.80	1.00	1.00	4.8
[P34]	1.00	1.00	0.20	0.00	0.50	2.7	[P69]	1.00	1.00	0.60	0.00	0.50	3.1
[P35]	1.00	1.00	0.00	0.00	0.50	2.5							

Appendix B – Chapter 6

B.1 Repeated Measures Design (RQ_{2.2})

Mantel r	TD Principal	Aspect of TD Interest	Project
r_{11}	Total TD Principal	<i>Coupling</i>	Project 1
r_{21}	Total TD Principal	<i>Cohesion</i>	Project 1
...	...	...	...
r_{51}	Total TD Principal	<i>Complexity</i>	Project 1
...	...	...	...
r_{1n}	Total TD Principal	<i>Coupling</i>	Project n
r_{2n}	Total TD Principal	<i>Cohesion</i>	Project n
...	...	...	Project n
r_{5n}	Total TD Principal	<i>Complexity</i>	Project n

B.2 Repeated Measures Design (RQ₃)

Mantel r	TD Interest	Aspect of TD Principal	Project
r_{11}	Total TD Interest	<i>Understandability</i>	Project 1
r_{21}	Total TD Interest	<i>Poorly Written Code</i>	Project 1
...	...	...	...
r_{51}	Total TD Interest	<i>Coding Standards</i>	Project 1
...	...	...	...
r_{1n}	Total TD Interest	<i>Understandability</i>	Project n
r_{2n}	Total TD Interest	<i>Poorly Written Code</i>	Project n
...	...	...	...
r_{5n}	Total TD Interest	<i>Coding Standards</i>	Project n

Appendix C – Chapter 7

C.1 Data Collection Instruments

1. Workshop Organization

Day	Activity	Time	Duration
1	Presentation-Introductions	09:00 – 09:15	15 minutes
	Interview of PM1 (RQ ₁)	09:15 – 09:40	25 minutes
	Interview of A1 (RQ ₁)	09:40 – 10:05	25 minutes
	Interview of PM2 (RQ ₁)	10:05 – 10:30	25 minutes
	Interview of A2 (RQ ₁)	10:30 – 10:55	25 minutes
	Break	10:55 – 11:15	20 minutes
	Focus Group on Practices (RQ ₁)	11:15 – 12:00	45 minutes

	Demonstration (RQ ₂)	12:00 – 12:20	20 minutes
	Questionnaire for CB Models (RQ ₂)	12:20 – 12:40	20 minutes
	Lunch Break	12:40 – 13:30	50 minutes
	Focus Group on Additional CB Models & Evaluation (RQ ₂)	13:30 – 15:00	90 minutes
2	Task Introduction	09:00 – 09:15	15 minutes
	Observation for PM1 (RQ ₃)	09:15 – 09:45	30 minutes
	Observation for A1 (RQ ₃)	09:45 – 10:15	30 minutes
	Observation for PM2 (RQ ₃)	10:15 – 10:45	30 minutes
	Observation for A2 (RQ ₃)	10:45 – 11:15	30 minutes
	Break	11:15 – 11:30	15 minutes
	Questionnaire (RQ ₃)	11:30 – 12:00	30 minutes
	Interview of PM1 (RQ ₃)	12:00 – 12:30	30 minutes
	Interview of A1 (RQ ₃)	12:30 – 13:00	30 minutes
	Interview of PM2 (RQ ₃)	13:00 – 13:30	30 minutes
	Interview of A2 (RQ ₃)	13:30 – 14:00	30 minutes

2. Interview Guide for RQ1

Block 0: (5min) Setting-up the Context

This set of questions does not contribute to the answer of RQ1. However, they are deemed as useful for setting up the context and warm-up the participants, by bringing into their mind some recent decisions that they have taken.

- What kind of decisions have you taken in the past?
- Have you taken quality optimizations decisions?
 - Can you please describe the decision?
 - Which QA was targeted?
 - How frequent are such decisions?
- Who took these decisions?
 - Have you used any tool for supporting the decision-making process?
 - Who introduced the decision request?

Block 1: (15min) Understanding the current decision-making practices

This set of questions will extract information on the current approach used by the company to relate decision-making about quality optimization. The questions will investigate different actions performed by different stakeholders and the tools used.

- Describe the current process, in terms of decision-making.
 - How do you gather relevant information?
 - How do you identify the alternatives?

How do you weigh the evidence?

How do you choose between the alternatives?

How do you take action?

Do you revisit decisions?

- What tools do you use for:

Handling evidence?

Evidence aggregation?

Evidence comparison?

Revisiting decisions?

Block 2: (5min) Investigating the benefits and limitations of the current decision-making practice

This set of questions will explore which are the positive and negative aspects of the practices used by the company at the moment.

- Which are the benefits and limitations of the current practice?

With which parts of the process are you satisfied?

What limitations do you experience?

In case respondents need motivation for their answers, we will refer back to the 6 steps of the previous question.

Block 3: (5min) Conclusion

- Thank you for your time.
- Explain next steps (transcribed, interview colleagues, analyze, results will be made available in an internal report and also in a publication)
- Ask if they want to receive the results by mail

3. Focus Group Guide for RQ1

Part 1 - Description of current practices (25mins)

- Quality Optimizations

How do you select QAs that will drive quality optimizations?

Univariate or Multivariate models?

How do you select the parameters that affect the decision?

How do you weigh them?

How do you aggregate them?

Do you mix sales, marketing and technical parameters?

Do you mix HW and SW parameters?

Is there a financial perspective in the analysis?

Is time framing for the decision considered?

How do you re-configure the parameters of the decision if it has to be taken in another time point?

Is there any means of visualization?

- Which are the stakeholders involved in the aforementioned process?

How do the actions of different stakeholders differ, in terms of the described process?

Who is responsible for each step?

Part 2 - Benefits and limitations of the current practices (20mins)

- What parts of the process are you satisfied form?
- What are the limitations?

4. Demonstration for RQ2

Cost to apply refactoring (P—Prevention Cost). This cost corresponds to the effort (directly related to amount of money) that CNET would be required to spend for transforming the code that runs in a CPU to a code that can run on GPU. In particular, we reuse the model developed by Marantos et al., which is able to predict the amount of code (in lines of code) that need to be re-written so as to transform C++ code (runs in CPU) to CUDA code (runs in GPU). Finally, to transform the lines of code to be maintained to the effort (in minutes) to make this transformation, we used bibliographic references that provide estimates for the cost of writing an individual line of code, including the associated reasoning, documentation, implementation and testing activities. At this point, we need to note that the specific equation comes as an illustrative example; in case a company holds internal data on the average effort for maintaining software, or the hourly cost of developers, it can use their own estimations rather than the default ones.

Cost for monitoring TD and Energy (A—Appraisal Costs). In order for CNET to monitor the levels of the two quality attributes of interest (namely energy efficiency and technical debt) we suppose that the company will use open-source solutions. Although these solutions come for free, some training needs to take place. Therefore, we calculate a training cost, by considering as parameters the number of employees to be trained and the hourly cost of the trainer.

Cost to maintain code (F—Failure Cost). Based on the same predictive model, we can observe that the maintainability of the CUDA code is usually worse than C++ code, since: (a) it is longer, and (b) it is more complex. Therefore, CNET needs to spend extra effort for software maintenance. Given this model, we predict the Cyclomatic Complexity of the CUDA code, and the Lines of Code of the CUDA version. To transform these metrics to maintenance costs, we reuse interest calculation from the BP methodology.

Benefit in terms of additional market share (B —Benefit). From any reduction in terms of energy, we suppose that the marketing department of CNET expects an increase in the number of customers. Therefore, we can calculate the benefit as a function of the expected new customers and the unit price. We note that since all the aforementioned costs have been calculated at a yearly basis, we calculate this benefit per year, as well.

Benefits in terms of updating existing maintenance contracts (B —Benefit). The improvement of the quality can be utilized as an increase in the price of the CNET product, since customers are expected to be willing to pay more for updating their existing installations. Therefore, this benefit can be calculated as the product of the increment of price and the number of customers that are willing to upgrade their subscription. For simplicity in this illustration, we do not consider possible contract cancelations. In case the manager wants to consider such potential losses, an additional cost model shall be added. Nevertheless, this is not necessary for the purpose of this task.

Benefit from Reduced Customer Support (B —Benefit). The fact that energy consumption is reduced, is expected to lead to less “low power” or “out-of-battery” failures. As a consequence, we expect less requests for the customer support of CNET. This benefit can be reflected as the product of expected decrease in support requests, average time to handle a request, and the average rate of the involved employees.

Cost Models

(P) Costs to re-write Code → From C++ to CUDA

- 200 Lines of Code to be transferred
- Cost per LoC Maintained takes 7.5 euros

(A) Costs for Measuring Energy and TD

- Training Costs
 - 8 hours of training the developers
 - instructor takes 50 euros per hour
- Both Solutions are OSS

(F) Costs in terms of Extra TD Interest

- in total 3000 lines of code updated every year
- the cyclomatic complexity will increase by 50%
- the code size will become 160% larger
- at least one part of the system is maintained with a probability 80%

Horizon 2020, Project No.780572 11/11/2019

Cost Models

Equation

(P) Costs to re-write Code → From C++ to CUDA

- 200 Lines of Code to be transferred
- Cost per LoC Maintained takes 7.5 euros

Cost per LoC * Lines of Code to be transferred

(A) Costs for Measuring Energy and TD

- Training Costs
 - 8 hours of training the developers
 - instructor takes 50 euros per hour
- Both Solutions are OSS

Training Sessions in Hours * Instructor Hourly Rate

(F) Costs in terms of Extra TD Interest

- in total 3000 lines of code updated every year
- the cyclomatic complexity increase by 50%
- the code size will become 160% larger
- at least one part of the system is maintained with a probability 80%

CC Increase * Lines of Code updated every year *
LoC Increase * Percentage of classes updated every year

Horizon 2020, Project No.780572

11/11/2019

Benefit Models

Direct Energy Gains

- kWh Saved per Day
- Days in Operation
- kWh Cost

Marketing Benefits

- Increased Market Share
 - an increase of 20% of active customers
- Upgrade of Existing Licenses
 - upgrade of 40 licenses
 - each one costing 50 euros
- Decrease Customer Support Costs
 - average hourly rate is 25 euros per hour
 - 80 hours of work will be saved

Horizon 2020, Project No.780572

11/11/2019

Benefit Models

Direct Energy Gains

- kWh Saved per Day
- Days in Operation
- kWh Cost

Equation

Marketing Benefits

- Increased Market Share
 - an increase of 20% of active customers
- Upgrade of Existing Licenses
 - upgrade of 40 licenses
 - each one costing 50 euros
- Decrease Customer Support Costs
 - average hourly rate is 25 euros per hour
 - 80 hours of work will be saved

Expected Increase * Market Share * Market Size * Price

Customers that Upgraded * Upgrade Cost

Customer Support Hourly Rate *
Hours saved while performing Customers' Support

Horizon 2020, Project No.780572 11/11/2019

5. Questionnaire for RQ2

Prevention Models

$$1. \text{ Hire Quality Personel} = \text{New_Employees} * \text{Yearly_Cost_of_Employee}$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$2. \text{ Perform Meetings} = \underbrace{(\text{Meeting_per_Month} * 12)}_{60} * \text{Number_of_Participants} * \text{Hourly_Rate} *$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

3. $Training = Default_Cost_of_Creation_of_Material + (Training_Sessions_in_Hours * Instructor_Hourly_Rate)$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

4. $Increase_Preventive_Maintenance_Effort = (Preventive_Maintenance_in_Hours * Testers_Hourly_Rate) + (Debugging_in_Hours * Developers_Hourly_Rate)$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy
----------------	-----------	---------	------	-----------

--	--	--	--	--

5. $Documentation = Number_of_Specifications_Required * Time_Needed_to_Develop_a_Document * Analyst_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

6. $Project_Planning = Additional_Time_for_Planning * Manager_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

7. $Prototyping = Number_of_Prototypes * Time_To_Build_a_Prototype * Developers_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

8. $Reuse_Proprietary_Software = Cost_of_Software + (Training_Sessions_in_Hours * Instructor_Hourly_Rate)$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

Appraisal Models

1. $Code_Inspection = (Number_of_Inspections_per_Month * 12) * Number_of_Participants * Developers_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$2. \quad \text{Design_Reviews} = (\text{Number_of_Reviews_per_Month} * 12) * \text{Number_of_Participants} * \text{Analysts_Hourly_Rate}$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$3. \quad \text{Unit_Testing} = \text{Number_of_Tests_in_Project} * \text{Time_To_Develop_a_Unit_Test} * \text{Developers_Hourly_Rate}$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$4. \quad \text{QA_Tools} = \text{Cost_of_Acquisition} + (\text{Training_Sessions_in_Hours} * \text{Instructor_Hourly_Rate})$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

--	--	--	--	--

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

5. $Quality_Audits = (Number_of_Audits_per_Month * 12) * Number_of_Participants * External_Expert_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

6. $Acceptance_and_Integration_Testing = Cost_to_Develop_Test_Cases + (Test_Cases_to_be_Executed * Time_Needed_to_Execute_Test * Testers_Hourly_Rate)$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$7. \quad \text{Metrics_Based_QA} = \text{Cost_to_Retrieve_Metrics} + (\text{Number_of_Metrics} * \text{Interpretation_Time} * \text{QA_Expert_Hourly_Rate})$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

$$8. \quad \text{Regression_Testing} = \text{Cost_to_Develop_Test_Cases} + (\text{Test_Cases_to_be_Executed} * \text{Time_Needed_to_Execute_Test} * \text{Testers_Hourly_Rate})$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

Failure Models

$$4. \quad \text{Necessary_Rework (e.g., Debugging, Quality Improvement)} = \text{Needed_Rework_in_Hours} * \text{Developers_Hourly_Rate}$$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

--	--	--	--	--

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

5. *Declining_Market_Positioning (e.g., poor quality, delay of release) = Lost_Customers * Price*

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

6. *Additional_Costs_for_Marketing = Extra_Hours_spent_in_Marketing_per_year * Marketeer_Hourly_Rate*

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy
----------------	-----------	---------	------	-----------

--	--	--	--	--

7. $Lost_Opportunity_Costs = Return_on_Best_Foregone_Option - Return_on_Chosen_Option$
To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

Benefit Models

1. $Increased\ Market\ Share = Expected\ Increase * Market\ Share * Market\ Size * Price$
To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

2. $Upgrade\ of\ Existing\ Licenses = Customers\ that\ Upgraded * Upgrade\ Cost$
To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very
-------------	--------	---------	------------	------

				Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

3. $Decreased_Need_for_Customer_Support = Hours_saved_while_performing_Customers'Support * Customer_Support_Hourly_Rate$

To what extent do you think that this model is accurate?

Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree

With what frequency do you think that model will be used?

Very Rarely	Rarely	Neutral	Frequently	Very Frequently

How easy is to assess the parameters needed to instantiate this model?

Very Difficult	Difficult	Neutral	Easy	Very Easy

6. Focus Group Guide for RQ2

Part 1 – Discussion on Default Models (30mins)

Regarding the following prevention cost model, related to the cost of hiring more employees in the QA section

$$Hire\ Quality\ Personel = New_Employees * Yearly_Cost_of_Employee$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$Perform\ Meetings = (Meeting_per_Month * 12) * Participants * Hourly_Rate * \left(\frac{Meeting_Duration_in_Minutes}{60} \right)$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Training} = \text{Creation_of_Material} + (\text{Training_Sessions_in_Hours} * \text{Instructor_Rate})$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Increase_Preventive_Maintenance_Effort} \\ = (\text{Preventive_Maintenance_in_Hours} * \text{Testers_Hourly_Rate}) \\ + (\text{Debugging_in_Hours} * \text{Developers_Hourly_Rate}) \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Documentation} = \text{Number_of_Specifications_Required} * \text{Time_Needed_to_Develop_a_Document} * \text{Analyst_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Project_Planning} = \text{Additional_Time_for_Planning} * \text{Manager_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Prototyping} = \text{Number_of_Prototypes} * \text{Time_To_Build_a_Prototype} * \text{Developers_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Reuse_Proprietary_Software} \\ = \text{Cost_of_Software} + (\text{Training_Sessions_in_Hours} * \text{Instructor_Hourly_Rate}) \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Code_Inspection} = (\text{Number_of_Inspections_per_Month} * 12) * \text{Number_of_Participants} * \text{Developers_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Design_Reviews} = (\text{Number_of_Reviews_per_Month} * 12) * \text{Number_of_Participants} * \text{Analysts_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Unit_Testing} = \text{Number_of_Tests_in_Project} * \text{Time_To_Develop_a_Unit_Test} * \text{Developers_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{QA_Tools} = \text{Cost_of_Acquisition} + (\text{Training_Sessions_in_Hours} * \text{Instructor_Hourly_Rate})$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Quality_Audits} = (\text{Number_of_Audits_per_Month} * 12) * \text{Number_of_Participants} * \text{External_Expert_Hourly_Rate}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Acceptance\&Integration_Testing} \\ = \text{Cost_to_Develop_Test_Cases} + (\text{Test_Cases_to_be_Executed} \\ * \text{Time_Needed_to_Execute_Test} * \text{Testers_Hourly_Rate}) \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Metrics_Based_QA} = \text{Cost_to_Retrieve_Metrics} + (\text{Number_of_Metrics} * \text{Interpretation_Time} * \text{QA_Expert_Hourly_Rate})$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Regression_Testing} = \text{Cost_to_Develop_Test_Cases} + (\text{Test_Cases_to_be_Executed} * \text{Time_Needed_to_Execute_Test} * \text{Testers_Hourly_Rate})$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Necessary_Rework (e.g., Debugging, Quality Improvement)} \\ = \text{Needed_Rework_in_Hours} * \text{Developers_Hourly_Rate} \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Declining_Market_Positioning (e.g., poor quality, delay of release)} = \text{Lost_Customers} * \text{Price}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Additional_Costs_for_Marketing} \\ = \text{Extr_Hours_spent_in_Marketing_per_year} * \text{Marketeer_Hourly_Rate} \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Lost_Opportunity_Costs} = \text{Return_on_Best_Foregone_Option} - \text{Return_on_Chosen_Option}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Increased_Market_Share} = \text{Expected_Increase} * \text{Market_Share} * \text{Market_Size} * \text{Price}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\text{Upgrade of Existing Licenses} = \text{Customers that Upgraded} * \text{Upgrade Cost}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

$$\begin{aligned} \text{Decreased_Need_for_Customer_Support} \\ = \text{Hours_saved_while_performing_Customers' Support} \\ * \text{Customer_Support_Hourly_Rat} \end{aligned}$$

- What additional parameters would you need?
- Would you use a different aggregation method?
- Do you agree with the weighting?

Part 2 – Proposal of Additional Models (30mins)

- What additional models would you need?
- Please define the model (parameters, aggregation method, weights)
- When would this model be used for?
- What stakeholders would need to be involved to instantiate the model?

Part 3 – Evaluation of the Proposed Approach (30mins)

- What is your overall impression of the proposed approach?
 - Usability?
 - Accuracy?
 - Timeliness?
- What is the generalizability of the approach?
 - Is it QA agnostic (e.g., energy, TD)?
- What is your commenting on the financial perspective that this approach offers?
 - Aggregation of parameters of different nature
 - Communication with management
 - Handle of trade-offs between QAs
 - Additional Complexity
 - Additional competences required
- How does the proposed approach compare to the current practices?
 - Benefits
 - Limitations

7. Task Description for RQ3

Suppose a system with the following structural characteristics:

- Lines of Code: 115000
- Coupling: 932
- Lack of Cohesion of Methods: 1252
- Polymorphism: 108

The average cost to fix a bug in this system is 120 \$. The system is on average maintained two times per month and the amount of change is on average 1200 lines of code. The customer support department comprises of 10 people and the yearly cost is 15000 \$.

Task 1 – Comparison of refactoring alternatives

Suppose that you have three alternatives for improving your system:

- a) Improve coupling by 10%
- b) Improve cohesion by 10%
- c) Improve polymorphism by 10%

For each alternative, four parameters are considered. The equations for cost and benefit models are provided below:

1. $\text{Cost_of_Refactoring} = \text{QA_Value} * \text{QA_Improvement} * \text{Refactoring_Weight}$
2. $\text{Benefits_from_Reducing_Cost_of_Maintenance} = \text{LoC}(\text{amount_of_change}) * \frac{\text{QA_Improvement}}{10} * \text{Maintenance_frequency_per_Year}$
3. $\text{Benefits_from_Reducing_Cost_of_Debugging} = \text{QA_Value} * \text{QA_Improvement} * \text{Fault_Proneness_Weight} * \text{Cost_of_Bug_Fixing}$
4. $\text{Benefits_in_Customers_Support_Expencies} = \text{Customer_Support_Cost} * \text{QA_Value} * \text{QA_Improvement} * \text{Fault_Proneness_Weight}$

The parameters of the equations are provided below or in the begging of this document:

QA	Refactoring Weight	Fault-Proneness Weight
coupling	0.802	0.011
Cohesion	0.017	0.007
polymorphism	2.642	0.028

Create three decisions in the ADMIT tool and calculate the profit or loss from each alternative.

Task 2 – Revisit Refactoring Decision

You are now asked to focus on a specific QA (Coupling). Investigate the deviation of profit or loss when improving coupling by **10 %**, **20%** and **50%**.

8. Interview Guide for RQ3

Block 1: (20min) Evaluation of the basic functional requirements of the tool

- How do you rate the following functionalities of the tool (in a scale from 1-10)?
 - Creation of a cost / benefit model
 - Creation of the equations
 - Visualization of the outcome
 - Revisit of a decision by altering the values of parameters
 - Customization of the tool / parameters of the equations
 - Default models

- Elaborate on the scores provided
- Is there any another feature of the tool (not listed here) you would like to comment about?
- Is there any additional feature that you would like to see in a future version of the tool?

Block 2: (10min) Evaluation of tool usability

- How do you perceive the usability of the system in terms of:
 - Effectiveness (i.e. the accuracy and completeness with which users achieve specified goals)
 - To what extend did you manage to complete the requested tasks?
 - How did you experience the navigability in the tool?
 - Efficiency (i.e. the resources expended in relation to the accuracy and completeness with which users achieve goals)
 - How much time did you spend per task? Was it more or less than the expected time needed based on your experience using the current process?
 - Satisfaction (the comfort and acceptability of use)
 - How confident did you feel while using the system?
 - Would you like to use this system frequently?
 - How complex would you characterize the system?
 - To what extend would you need the support to be able to use the system?
 - How easy would you consider to learn how to use the tool?
 - What kind of previous knowledge/ background would you consider as mandatory before you could get going with the system?
 - How would you describe the experience in terms of the reactions of the system to possible stimuli (e.g., mouse clicks or data entry)?
 - Have you faced any other usability issues?

Block 3: (5min) Conclusion

- Thank you for your time.
- Explain next steps (transcribed, interview colleagues, analyze, results will be made available in an internal report and also in a publication)
- Ask if they want to receive the results by mail

C.2 Default Cost-Benefit Models

Default Prevention Costs

Costs for Hiring Quality Personnel = New Employees * Cost of Employee

Costs for Performing Meetings = $\frac{\text{Number of Meetings} * \text{Number of Participants} * \text{Hourly Rate}}{(\text{Meeting Duration in Minutes} / 60)}$

Costs of Training = Creation of Material + (Training Sessions in Hours * Instructor Hourly Rate)
(Preventive Maintenance in Hours * Testers Hourly Rate) +

Increase Preventive Maintenance Effort = (Debugging in Hours * Developers Hourly Rate)
Number of Specifications Required * Time Needed to Develop a Document *

Cost of Producing Documentation = Analyst Hourly Rate

Increase Cost for Project Planning = Additional Time for Planning * Manager Hourly Rate

Cost of Prototyping = Number of Prototypes * Time to Build a Prototype * Developers Hourly Rate

Cost for Reusing Proprietary Software = Cost of Software + (Training Sessions in Hours * Instructor Hourly Rate)

#Standards to be Applied + (Training in Hours * Instructor Hourly Rate) +

Costs for Enabling the Use of Standards = (Time for Standard Compliance * Company Hourly Rate)

Default Appraisal Costs

Cost for Performing Code Inspection = $\frac{(\text{Number of Inspections per Month} * 12) * \text{Number of Participants} * \text{Developers Hourly Rate}}{(\text{Number of Reviews per Month} * 12) * \text{Number of Participants} *}$

Cost for Performing Design Reviews = Analysts Hourly Rate
Number of Tests in Project * Time to Develop a Unit Test *

Cost for Performing Unit Testing = Developers Hourly Rate

Cost for Using Quality Assurance Tools = Cost of Acquisition + (Training Sessions in Hours * Instructor Hourly Rate)

Cost for Performing Quality Audits = $\frac{(\text{Number of Audits per Month} * 12) * \text{Number of Participants} * \text{External Expert Hourly Rate}}{\text{Cost to Develop Test Cases} + (\text{Test Cases to be Executed} * \text{Time Needed to Execute Test} * \text{Testers Hourly Rate})}$

Cost for Performing Acceptance / Integration Testing = $\frac{\text{Cost to Retrieve Metrics} + (\text{Number of Metrics} * \text{Interpretation Time} * \text{QA Expert Hourly Rate})}{\text{Cost to Develop Test Cases} + (\text{Test Cases to be Executed} * \text{Time Needed to Execute Test} * \text{Testers Hourly Rate})}$

Cost for Performing Metrics-Based Quality Assurance = $\frac{\text{Cost to Develop Test Cases} + (\text{Test Cases to be Executed} * \text{Time Needed to Execute Test} * \text{Testers Hourly Rate})}{\text{Time Needed to Execute Test} * \text{Testers Hourly Rate}}$

Default Failure Costs

Necessary Rework = Needed Rework in Hours * Developers Hourly Rate

Declining Market Positioning = Lost Customers * Product/Service Price

Additional Costs for Marketing = FTE in Marketing * Marketeer Hourly Rate

Lost Opportunity Costs = Return on Best Foregone Option - Return on Chosen Option

Default Benefits

Increased Market Share = (New Customers - Old Customers) * Price

Upgrade of Existing Licenses = Customers that Upgrade * Upgrade Cost

Decreased Need for Customer Support = Hours Saved from Customer Support * Customer Support Hourly Rate

References

- 1061-1998: IEEE Standard for a Software Quality Metrics Methodology, IEEE Standards, IEEE Computer Society, reaffirmed Dec. 2009.
- Allman, E., “Managing technical debt”, *Communication*, 55 (5), ACM, pp. 50-55, 2012.
- 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, pp.100–121, 2016.
- Alves, V., Niu, N., Alves C., and Valenca, G., “Requirements engineering for software product lines: a systematic literature review”, *Information and Software Technology*, Elsevier, 52 (8), pp. 806–820, August 2010.
- Ampatzoglou A., Gkortzis, A., Charalampidou, S., and Avgeriou P., “An embedded multiple-case study on OSS design quality assessment across domains”, *Proceedings of the 7th International Symposium on Empirical Software Engineering and Measurement (ESEM '13)*, ACM/IEEE, Baltimore, MA, USA, pp. 255-258, 2013.
- Ampatzoglou, Ar., Ampatzoglou, Ap., Chatzigeorgiou, A., and Avgeriou, P., “The financial aspect of managing technical debt: A systematic literature review,” *Information and Software Technology*, Elsevier, 64, pp. 52–73, 2015a.
- Ampatzoglou, Ar., Ampatzoglou, Ap., Avgeriou, P., and Chatzigeorgiou, A., “Establishing a framework for managing interest in technical debt”, *5th International Symposium on Business Modeling and Software Design (BMSD '15)*, Milan, Italy, 2015b.
- Ampatzoglou, Ar., Ampatzoglou, Ap., Chatzigeorgiou, A., and Avgeriou, P., Abrahamsson, P., Martini, A., Zdun, U., and Systa, K., “The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study”, *8th International Workshop on Managing Technical Debt (MTD '16)*, IEEE, Raleigh, NC, USA, pp. 9-16, 2016a.
- Ampatzoglou, Ar., Ampatzoglou, Ap., Avgeriou, P., and Chatzigeorgiou, A., “A Financial Approach for Managing Interest in Technical Debt”, *LNBIP*, Springer, 257, pp. 117-133, 2016b.
- Ampatzoglou, Ar., Michailidis, A., Sarikyriakidis, C., Ampatzoglou, Ap., Chatzigeorgiou, A., and Avgeriou, P., “A framework for managing interest in technical debt: an industrial validation”, *Proceedings of the 2018 International Conference on Technical Debt (TechDebt 2018)*, ACM, Gothenburg, Sweden, pp. 115-124, May 2018.
- Ampatzoglou, Ar., Mittas, N., Tsintzira, A. A., Ampatzoglou, Ap., Arvanitou, E. M., Chatzigeorgiou, A., Avgeriou, P., Angelis, L., “Exploring the Relation between

- Technical Debt Principal and Interest: An Empirical Approach”, *Information and Software Technology*, 128, p. 106391, 2020a.
- Ampatzoglou, Ar., Arvanitou, E. M., Ampatzoglou, Ap., Avgeriou, P., Tsintzira, A. A., and Chatzigeorgiou, A., “Architectural Decision-making as a Financial Investment: An Industrial Case Study”, *Information and Software Technology*, in press, p. 106412, 2020b.
- Arvanitou, E. M., Ampatzoglou, A., Chatzigeorgiou, A., and Avgeriou, P., “Software metrics fluctuation: a property for assisting the metric selection process”, *Information and Software Technology*, 72, pp.110-124, 2016.
- Arvanitou, E. M., Ampatzoglou, A., Bibi, S., Chatzigeorgiou, A., and Stamelos, I., “Monitoring Technical Debt in an Industrial Setting”, *Proceedings of the Evaluation and Assessment on Software Engineering*, pp. 123-132, 2019.
- Avgeriou, P., Kruchten, P., Ozkaya, I., and Seaman, C., “Managing technical debt in software engineering (Dagstuhl seminar 16162)”, *Dagstuhl Reports*, 6 (4), Schloss Dagstuhl-Leibniz-Zentrum für Informatik, 2016.
- Babar M. A. and Gorton I., “A tool for managing software architecture knowledge”, *2nd Workshop on Sharing and Reusing Architectural Knowledge-Architecture, Rationale, and Design Intent (SHARK/ ADI'07: ICSE Workshops 2007)*, IEEE, pp. 11-11, 2007
- Basili, V., Caldiera, G., and Rombach, D., “The Goal Question Metric Approach”, *Encyclopedia of Software Engineering*, John Wiley & Sons, pp. 528-532, 1994.
- Bass, L., Clements, P., and Kazman, R., “Software Architecture in Practice”, Addison-Wesley Professional, 2nd edition, 2003.
- Benaroch, M., “Managing Information Technology Investment Risk: A Real Options Perspective”, *Journal of Management Information Systems*, ACM, 19 (2), pp. 43-84, October 2002.
- Besker, T., Martini, A., and Bosch, J., “The Pricey Bill of Technical Debt: When and by Whom will it be Paid?”, *33rd IEEE International Conference on Software Maintenance and Evolution (ICSME)*, IEEE, Shangai, China, pp. 13-23, Sept. 2017.
- Boehm, B. W. and Sullivan, K. J. “Software economics: a roadmap”, *22nd International Conference on the Software Engineering (ICSE '00)*, ACM, pp. 319-343, Limerick, Ireland, 4-11 June 2000.
- Borison, A., “Real Options Analysis: Where are the Emperor’s Clothes?”, *Journal of Applied Corporate Finance*, Wiley & Sons, 17 (2), pp. 17-31, June 2005.

- Bowersox, D. J., Stank, T. P., and Daugherty, P. J., "Lean launch: managing product introduction risk through response-based logistics," *Journal of Product Innovation Management*, 16 (6), pp. 557–568, 1999.
- Brooke J., "SUS-A quick and dirty usability scale". *Usability evaluation in industry*, London, 189 (194), pp. 4-7. 1996.
- Buschmann, F., "To Pay or Not to Pay Technical Debt", *IEEE Software*, 28 (6), IEEE Computer Society, pp. 29-31, 2011.
- Capilla R., Jansen A., Tang A., Avgeriou P., Ali Babar M., "10 years of Software Architecture Knowledge Management: Practice and Future", *Journal of Systems and Software*, Elsevier, 116, pp. 191–205, 2016.
- Capilla R., Nava F., Montes J., Carrillo C., "ADDSS: Architecture Design Decision Support System Tool", *Proceedings of the 23rd IEEE/ACM International Conference on Automated Software Engineering*, IEEE, pp. 487–488, 2008.
- Chapin, N. J., Hale, E., Khan, K., Ramil, J. F., and Tan, W. G., "Types of software evolution and software maintenance", *Journal of Software Maintenance and Evolution: Research and Practice*, Wiley & Sons, 13 (1), pp. 3 – 30, 2001.
- Chatzigeorgiou, A., Ampatzoglou, Ap, Ampatzoglou, Ar., and Amanatidis, T., "Estimating the Breaking Point for Technical Debt", *7th International Workshop on Managing Technical Debt (MTD '15)*, IEEE Computer Society, 2015.
- Chidamber, S. R., Darcy, D. P., and Kemerer, C. F., "Managerial Use of Metrics for Object Oriented Software: An Exploratory Analysis", *Transactions on Software Engineering*, IEEE Computer Society, 24 (8), pp. 629-639, Aug. 1998.
- Chin, S., Huddleston, E., Bodwell, W., Gat, I., "The Economics of Technical Debt", *Cutter IT Journal*, 2010.
- Clements P., Bass L., "Relating Business Goals to Architecturally Significant Requirements for Software Systems", *CMU Journal*, 2018
- Codabux, Z. and Williams, B., "Managing technical debt: An industrial case study," *4th International Workshop on Managing Technical Debt (MTD' 13)*, pp. 8–15, 2013.
- Conejero, J. M., Rodríguez-Echeverría, R., Hernández, J., Clemente, P. J., Ortiz-Caraballo, C., Jurado, E., and Sánchez-Figueroa, F., "Early evaluation of technical debt impact on maintainability", *Journal of Systems and Software*, Elsevier, 142, pp. 92-114, 2018.
- Cunningham, W., "The WyCash Portfolio Management System", *7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '92)*, pp. 29-30, Vancouver, Canada, 5-10 October 1992.

- Curtis, B., Sappidi, J., and Szynekarski, A., “Estimating the size, cost, and types of Technical Debt”, *3rd International Workshop on Managing Technical Debt (MTD ’12)*, IEEE, Zurich, Switzerland, pp. 49 – 53, 2012.
- de Almeida R. R., Kulesza U., Treude C., and Lima A. H. G., “Aligning Technical Debt Prioritization with Business Objectives: A Multiple-Case Study”, *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME’18)*, IEEE, pp. 655-664, 2018
- Del Rosso, C., “Software performance tuning of software product family architectures: Two case studies in the real-time embedded systems domain,” *Journal of Systems and Software*, Elsevier, 81(1) pp. 1–19, Jan. 2008.
- De Winter, J. F. C. and Dodou, D., “Five-Point Likert Items: t test versus Mann-Whitney-Wilcoxon”, *Practical Assessment, Research, and Evaluation*, 15(1), p. 11, 2010.
- Di Penta, M., “Combining quantitative and qualitative methods (when mining software data)”, *Perspectives on Data Science for Software Engineering*, Morgan Kaufmann, pp. 205-211, 2016.
- Duffie, D. and Singleton, K. J., “Credit Risk Pricing, Measurement, and Management”, Princeton University Press, 2003.
- Dybå, T. and Dingsoyr, T., “Empirical studies of agile software development: A systematic review”, *Information and Software Technology*, Elsevier, 50 (9), pp. 833-859, August 2008.
- Eisenberg, R. J., “A threshold-based approach to technical debt”, *ACM SIGSOFT Software Engineering Notes*, 37 (2), ACM, pp. 1 - 6, 2012.
- Eisenberg, R., “Management of Technical Debt: A Lockheed Martin Experience Report”, *5th International Workshop on Managing Technical Debt (MTD’ 13)*, Baltimore, USA, 9 October 2013.
- Elizondo-Noriega A., Güemes-Castorena, D., and Beruvides M., “An analysis on the prevention, appraisal, and failure model in COQ: Convergences and contradictions”, *IIE Annual Conference*, pp. 758-763, 2017.
- Erdogmus, H., “Comparative evaluation of software development strategies based on Net Present Value”, *1st Workshop on Economics Driven Software Engineering Research (EDSER’ 99)*, Los Angeles, USA, 17 May 1999.
- Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., and Gorton, I., “Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt,” *10th Joint Meeting on Foundations of Software Engineering*, New York, NY, USA, 2015, pp. 50–60, 2015.

- Ernst, N., “On the role of requirements in understanding and managing technical debt”, *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE, 2012.
- Evans, J. D., “Straightforward Statistics for the Behavioral Sciences”, Brooks/Cole Publishing, Pacific Grove, California, USA, 1996.
- Farris, P.W., Bendle, N.T., Pfeifer P.E., and Reibstein, D.J., “Marketing Metrics: The Definitive Guide to Measuring Marketing Performance”, Wharton School Publishing, 2nd Edition, 2010.
- Feigenbaum A.V., “Total Quality Control”, *Harvard Business Review*, 34 (6), pp. 93-101, 1956.
- Feigenbaum A.V., “Total Quality Control”, McGraw-Hill, New York, 1961.
- Feitosa, D., Ampatzoglou, A., Avgeriou, P., and Nakagawa, E. Y., “Investigating Quality Trade-offs in Open Source Critical Embedded Systems”, *Quality of Software Architectures (QoSA' 15)*, 2015.
- Field, A., “Discovering Statistics using IBM SPSS Statistics”, SAGE Publications Ltd., 2013.
- Fowler, M., “Refactoring: improving the design of existing code”, Addison-Wesley, 2018.
- Friedman, M., “Factors affecting the level of interest rate”, *Conference on Savings and Residential Financing*, USLL, pp. 11-27, Chicago, USA, 1968.
- García-Valls, M., Escribano-Barreno, J. and García-Muñoz J., “An extensible collaborative framework for monitoring software quality in critical systems”, *Information and Software Technology*, Elsevier, 107, pp. 3-17, 2019.
- Grubb, P. and Takang, A., “Software Maintenance: Concepts and Practice”, 2nd Edition, World Scientific Publishing Company, 2003.
- Guo, Y. and Seaman, C., “A portfolio approach to technical debt management”, *2nd International Workshop on Managing Technical Debt*, ACM, 2011.
- Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., da Silva, F., Santos, A.L., Siebra, C., “Tracking technical debt - An exploratory case study”, *27th International Conference on Software Maintenance (ICSM '11)*, IEEE Computer Society, 2011.
- Harman, M., “The current state and future of search-based software engineering.”, *Future of Software Engineering (FOSE'07)*, pp. 342-357. IEEE, 2007.
- Harrington, H. J., “Poor-Quality Cost: Implementing, Understanding, and Using the Cost of Poor Quality”, *CRC Press*, 11, 1987.
- Hevner, A. R., “A three cycle view of design science research”, *Scandinavian journal of information systems*, 19 (2), p4, 2007.

- Hofmeister C., Kruchten P., Nord R. L., Obbink H., Ran A., and America P., "A general model of software architecture design derived from five industrial approaches", *Journal of Systems and Software*, Elsevier, 80 (1), January 2007
- Houkes, W., "The open border: two cases of concept transfer from organisms to artifacts", *Functions in biological and artificial worlds: comparative philosophical perspectives*, MIT Press, pp. 227-242, 2009.
- Hull, T., "Options, Futures and Other Derivatives", Pearson Prentice Hall, 7th Edition, 2009.
- ISO/IEC 9126-1:2001, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland, 2001.
- ISO/IEC/IEEE 14764-2006 Standard for Software Engineering - Software Life Cycle Processes - Maintenance, IEEE Computer Society, 18 September 2006.
- Ivarsson, M. and Gorschek, T., "A Method for Evaluating Rigor and Industrial Relevance of Technology Evaluations," *Empirical Software Engineering*, Springer, 16 (3), pp. 365-395, June 2011.
- Izurieta, C., Griffith, I., Reimanis, D., Luhr, R., "On the uncertainty of technical debt measurements", *Proceedings of the 4th International Conference on Information Science and Applications (ICISA '13)*, IEEE, pp.1-4, Suwon, South Korea, 2013.
- Juran J.M., "Quality Control Handbook", McGraw-Hill, 1st ed., New York, 1951.
- Karg L. M., Grottke M., and Beckhaus A., "A systematic literature review of software quality cost research", *Journal of Systems and Software*, Elsevier, 84(3), pp. 415-427, 2011.
- Kazman R., Asundi J., and Klein M., "Quantifying the costs and benefits of architectural decisions", *Proceedings of the 23rd International Conference on Software Engineering (ICSE 2001)*, IEEE, pp. 297-306, 2001.
- Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyevev, S., Fedak, V., and Shapochka, A., "A case study in locating the architectural roots of technical debt", *37th International Conference on Software Engineering (ICSE)*, IEEE/ACM, 2, pp. 179-188, Florence, Italy, May 2015.
- Kellison, S., "The theory of interest", McGraw-Hill, 3rd Edition, 2008.
- Keung, J. W., Kitchenham, B. A., and Jeffery, D. R., "Analogy-X: Providing statistical inference to analogy-based software cost estimation", *Transactions on Software Engineering*, IEEE, 34(4), pp. 471-484, 23 May 2008.
- Kim S., and Nakhai B., "The dynamics of quality costs in continuous improvement", *International Journal of Quality and Reliability Management*, Emerald, 25(8), pp.842-859, 2008

- Kitchenham, B. A., Budgen, D., Brereton, O. P., “Using mapping studies as the basis for further research: A participant-observer case study”, *Information and Software Technology*, Elsevier, 53(6), pp. 638-651, June 2011.
- Kitchenham, B. and Pfleeger, S. L., “Principles of Survey Research Part 2: Designing a survey”, *Special Interest Group on Software*, ACM, 27 (1), pp. 18-20, January 2002.
- 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*, Elsevier, 51 (1), pp 7-15, January 2009.
- Kontio J., Bragge J., Lehtola L, “The focus group method as an empirical tool in software engineering”, *Guide to Advanced Empirical Software Engineering*, Springer-Verlag, pp. 93-116, 2007
- Kosti, M. V., Ampatzoglou, A., Chatzigeorgiou, A., Pallas, G., Stamelos, I., and Angelis, L., “Technical Debt Principal Assessment Through Structural Metrics”, *43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, Vienna, Austria, 2017.
- Kruchten P., “What do software architects really do?”, *Journal of Systems and Software*, Elsevier, 81(12), pp. 2413-2416, 2008.
- Kruchten P., Lago P., and van Vliet H., “Building up and reasoning about architectural knowledge”, *International Conference on the Quality of Software Architectures*, Springer, pp. 43-58, Berlin, Heidelberg, 2006.
- Kruchten P., Lago P., Van Vliet H., and Wolf T., “Building up and exploiting architectural knowledge”, *5th Working IEEE/IFIP Conference on Software Architecture (WICSA'05)*, IEEE, pp. 291-292, 2005.
- Kruchten, P., “The Rational Unified Process: An introduction”, Addison-Wesley Longman Publishing, 2nd Edition, 2000.
- Kruchten, P., Nord, R. L., and Ozkaya, I., “Technical Debt: From Metaphor to Theory and Practice”, *IEEE Software*, IEEE Computer Society, 29 (6), pp. 18-21, June 2012.
- Lee J., Kang S., and Kim C. K., “Software architecture evaluation methods based on cost benefit analysis and quantitative decision-making”, *Empirical Software Engineering*, Springer, 14 (4), pp. 453-475, 2009.
- Lethbridge T. C., Sim S. E., and Singer J., “Studying software engineers: Data collection techniques for software field studies”, *Empirical software engineering*, 10 (3), pp. 311-341, 2005.

- Letouzey, J. L. and Ilkiewicz, M., “Managing Technical Debt with the SQALE Method”, *IEEE Software*, IEEE, 29 (6), pp. 44–51, 2012.
- Letouzey, J. L. and T. Coq., “The sqale analysis model: An analysis model compliant with the representation condition for assessing the quality of software source code”, *2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID '10)*, IEEE, Nice, France, pp. 43-48, 2010.
- Letouzey, J. L., “The sqale method for evaluating technical debt”, *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, 2012.
- Li, W. and Henry, S., “Object-oriented metric that predict maintainability”, *Journal of Systems and Software*, Elsevier, 23 (2), pp. 111-122, November 1993.
- Li, Z., Avgeriou, P., and Liang, P., “A systematic mapping study on technical debt and its management”, *Journal of Systems and Software*, Elsevier, 101, pp. 193-220, March 2015.
- Li, Z., Liang, P., and Avgeriou, P., “Architectural Debt Management in Value-oriented Architecting”, *Economics-Driven Software Architecture*, Elsevier, pp. 183–204, 2014.
- Liang P., Jansen A., and Avgeriou P., “Collaborative software architecting through knowledge sharing”, *Collaborative Software Engineering*, Springer, pp. 343-367, Berlin, Heidelberg, 2010.
- Lim, E., Taksande, N., and Seaman, C., “A Balancing Act: What Software Practitioners Have to Say About Technical Debt,” *IEEE Software*, IEEE Computer Society, 29, (6), pp. 22–27, Nov. 2012.
- MacCormack, A., and Sturtevant, D. J., “Technical debt and system architecture: The impact of coupling on defect-related activity”, *Journal of Systems and Software*, Elsevier, 120, pp. 170-182, 2016.
- Mallick, D. N. and Schroeder, R. G., “An Integrated Framework for Measuring Product Development Performance in High Technology Industries,” *Production Operation and Management*, 14, (2), pp. 142–158, Jun. 2005.
- Mantel N., “The detection of disease clustering and a generalized regression approach”, *Cancer research*, 27 (2), pp.209-220, 1967.
- Manteuffel C., Tofan D., Avgeriou P., Koziolk H., and Goldschmidt T., “Decision architect—A decision documentation tool for industry”, *Journal of Systems and Software*, Elsevier, 112, pp. 181-198, 2016.
- Martin, R. C., “Agile software development: principles, patterns, and practices”, Prentice Hall PTR, 2003.

- Martini, A. and Bosch, J., "Towards prioritizing Architecture Technical Debt: information needs of architects and product owners," *41st Euromicro SEAA Conference*, Funchal, Madeira, August 2015.
- 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*, Elsevier, 163, pp. 42-61, 2018.
- Martini, A., Bosch, J., and Chaudron, M., "Investigating Architectural Technical Debt Accumulation and Refactoring over Time: a Multiple-Case Study," *Information and Software Technology*, July 2015.
- Mavridis, A., Ampatzoglou, A., Stamelos, I., Sfetsos, P., and Deligiannis, I., "Selecting Refactorings: An Option Based Approach", *8th International Conference on Quality of Information and Communications Technology (QUATIC' 12)*, IEEE Computer Society, pp. 272-277, Lisbon, Portugal, 3-6 September 2012.
- McGregor, J. D., Monteith, J., Zhang, J., "Technical debt aggregation in ecosystems", *3rd International Workshop on Managing Technical Debt (MTD '12)*, IEEE Computer Society, 2012.
- Menzies, T., "Correlation is not causation (or, when not to scream "Eureka!")", *Perspectives on Data Science for Software Engineering*, Morgan Kaufmann, pp. 327-330, 2016.
- Mens, T., Claes, M., Grosjean, P., and Serebrenik, A., "Studying evolving software ecosystems based on ecological models", *Evolving software systems*, Springer, pp. 297-326, Berlin, Heidelberg, 2014.
- Mishkin, F., and Eakins, S., "Financial Markets and Institutions", Pearson Prentice Hall, 7th Edition, 2012.
- Misra, S. H., "Modeling design/coding factors that drive maintainability of software systems", *Software Quality Journal*, Springer 13(3), pp. 297-320, 2005.
- Nickerson, R., Muntermann, J., Varshney, U., and Isaac, H., "Taxonomy development in information systems: developing a taxonomy of mobile applications", *17th European Conference in Information Systems (ECIS '09)*, pp. 1138–1149, Italy, 8 – 10 June 2009.
- Nord, R., Ozkaya, I., Kruchten, P., Gonzalez-Rojas, M., "In Search of a Metric for Managing Architectural Technical Debt", *2012 Joint Working IEEE/IFIP Conference on Software Architecture (WICSA) and European Conference on Software Architecture (ECSA)*, IEEE Computer Society, 2012.
- Nowak M. and Pautasso C., "Team situational awareness and architectural decision-making with the software architecture warehouse", *Proceedings of the 7th European*

- Conference (ECSA)*, Montpellier, France, *Volume 7957 of Lecture Notes in Computer Science*, Springer, pp.146–161, Berlin, Heidelberg, 2013.
- Nugroho, A., Visser, J., and Kuipers, T., “An empirical model of technical debt and interest”, *2nd International Workshop on Managing Technical Debt (MTD’ 11)*, ACM, pp. 1 - 8, Hawaii, USA, May 2011.
- O’ Keeffe, M. and Cinnéide, M. O., “Search-based refactoring for software maintenance”, *Journal of Systems and Software*, 81 (4), pp. 502-516, 2008.
- Ogawa, R.T., and Malen, B., “Towards rigor in reviews of multivocal literatures: applying the exploratory case study method”, *Review of Educational Research*, SAGE Publications, 61 (3), pp. 265–286, Fall 1991.
- Oliveira, M.F.S., Redin, R.M., Carro, L., Lamb, L., and Wagner, F., “Software Quality Metrics and their Impact on Embedded Software,” *5th International Workshop on Model-based Methodologies for Pervasive and Embedded Software, (MOMPES’08)*, pp. 68–77, 2008.
- Ouni, A., Kessentini, M., Sahraoui, H., Inoue, K., and Deb, K., “Multi-criteria code refactoring using search-based software engineering: An industrial case study’, *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25 (3), pp. 1-53, 2016.
- Parnas, D. L., “Software Aging”, *6th International Conference on Software Engineering (ICSE ‘94)*, IEEE Computer Society, pp. 279-287, Sorrento, Italy, 16 -21 May 1994.
- Peimbert-García R. E., Limón-Robles J., Beruvides M.G., and García-Hernández D., “An economic framework for total productive maintenance (TPM)”. *Proceedings of 62nd IIE Annual Conference and Expo*, Institute of Industrial and Systems Engineers (IISE), pp. 2760-2769, 2012.
- Pinheiro, J. and Bates, D., “Mixed-effects models in s and s-plus”, Springer Science and Business, 2006.
- Plunkett J. J. and Dale B. G., “A review of the literature on quality-related costs”, *International Journal of Quality & Reliability Management*, Emerald, 4(1), pp. 40-52, 1987.
- Poort E. and van Vliet H., "RCDA: Architecting as a risk- and cost management discipline", *Journal of Systems and Software*, Elsevier, 85(9), pp.1995-2013, 2012.
- Porter L. J. and Rayner P., “Quality costing for total quality management”, *International Journal of Production Economics*, Elsevier, 27(1), pp.69-81, 1992.
- Rauscher, T. G. and Smith, P. G., “From experience time-driven development of software in manufactured goods,” *Journal of Production and Innovation Management*, 12 (3), pp. 186–199, Jun. 1995.

- Reilly, F. and Brown, K., “Investment Analysis and Portfolio Management”, South-Western Cengage Learning, 10th Edition, 2012.
- Riaz, M., Mendes, E., and Tempero, E., “A systematic review of software maintainability prediction and metrics”, *Proceedings of the 3rd International Symposium on Empirical Software Engineering and Measurement*, IEEE, Florida, USA, pp. 367-377, 2009.
- Rovegård, P., Angelis, L., and Wohlin, C., “An empirical study on views of importance of change impact analysis issues”, *Transactions on Software Engineering*, IEEE, 34(4), pp. 516-530, 2008.
- Runeson, P. M. Host, A. Rainer, and B. Regnell, “Case Study Research in Software Engineering: Guidelines and Examples”, John Wiley & Sons, 2012.
- Schmid, K., “On the limits of the technical debt metaphor some guidance on going beyond”, *4th International Workshop on Managing Technical Debt (MTD '13)*, IEEE Computer Society, pp. 63 - 66, San Francisco, USA, 18 - 26 May 2013.
- Schnappinger, M., Osman, M.H., Pretschner, A., and Fietzke, A., “Learning a classifier for prediction of maintainability based on static analysis tools”, *Proceedings of the 27th International Conference on Program Comprehension*, IEEE Press, pp. 243-248, 2019.
- Seaman, C., Guo, Y., “Measuring and monitoring technical debt”, *Advances in Computers*, 82, pp. 25-46, Elsevier 2011.
- Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., Vetró, A., “Using technical debt data in decision making: Potential decision approaches”, *3rd International Workshop on Managing Technical Debt (MTD' 12)*, IEEE Computer Society, 2012.
- Siebra, C. S., Tonin, G. S., Silva, F. Q., Oliveira, R. G., Junior, A. L., Miranda, R. C., Santos, A. L., “Managing technical debt in practice: an industrial report”, *6th International Symposium on Empirical Software Engineering and Measurement (ESEM)*, ACM, 2012.
- Snipes, W., Robinson, B., Guo, Y., Seaman, C., “Defining the decision factors for managing defects: A technical debt perspective”, *3rd International Workshop on Managing Technical Debt (MTD' 12)*, IEEE Computer Society, 2012.
- Sommerville, I., “Software Engineering”, 9th Edition, Addison-Wesley, 2010.
- SonarQube, “SonarQube Documentation”, <https://docs.sonarqube.org/display/>, May 2018.
- Spencer D., “Card Sorting: Designing Usable Categories”, Rosenfeld Media, 1st edition, April 2009.

- Stroustrup, B., "Abstraction and the C++ Machine Model," *Embedded Software and Systems*, Springer, Berlin Heidelberg, pp. 1–13, 2005.
- Tang A., Avgeriou P., Jansen A., Capilla R., Ali Babar M., "A Comparative Study of Architecture Knowledge Management Tools", *Journal of Systems and Software*, Elsevier, 83(3), pp. 352-370, 2010.
- Tang A., Jin Y., and Han J., "A rationale-based architecture model for design traceability and reasoning", *Journal of Systems and Software*, Elsevier, 80(6), pp. 918-93, 2007.
- Tichy, W. F., and Padberg, F., "Empirical methods in software engineering research", *29th International Conference on Software Engineering (ICSE'07 Companion)*, IEEE, pp. 163-164, 2007.
- Tom, E., Aurum, A., and Vidgen, R., "An exploration of technical debt", *Journal of Systems and Software*, Elsevier, 86 (6), pp. 1498-1516, June 2013.
- Tsantalis, N., Chaikalis, T., and Chatzigeorgiou, A., "JDeodorant: Identification and removal of type-checking bad smells", *2008 12th European Conference on Software Maintenance and Reengineering*, IEEE, pp. 329-331, 2008.
- Tsintzira, A. A., Ampatzoglou, Ar., Matei, O., Ampatzoglou, Ap., Chatzigeorgiou, A., and Heb, R., "Technical Debt Quantification through Metrics: An Industrial Validation", *15th China-Europe International Symposium on Software Engineering Education (CEISEE' 19)*, IEEE TEMS, Lisbon-Caparica, Portugal, May 2019.
- van Heesch U., Avgeriou P., and Hilliard R., "A documentation framework for architecture decisions", *Journal of Systems and Software*, 85 (4), Elsevier, pp. 795-820, 2012.
- van Koten, C. and Gray, A., "An application of Bayesian network for predicting object-oriented software maintainability", *Information and Software Technology*, Elsevier, 48 (1), pp. 59-67, 2006.
- van Solingen R., Basili V., Caldiera G., and Rombach H. D., "Goal Question Metric (GQM) Approach", *Encyclopedia of Software Engineering*, John Wiley & Sons, 2002.
- van Vliet H., and Tang A., "Decision-making in software architecture", *Journal of Systems and Software*, Elsevier, 117, pp. 638-644, 2016.
- van Vliet, H. "Software Engineering: Principles and Practice", John Wiley & Sons, 2008.
- Wake, W. C., "Refactoring Workbook", Addison-Wesley Professional, 1st Edition, 2003.
- Wieringa, R., "Design science as nested problem solving", *Proceedings of the 4th international conference on design science research in information systems and technology*, pp. 1-12, 2009.
- Wohlin, C., Runeson, P., Höst, M., Ohlsson, M. C., Regnell, B., and Wesslén, A., "Experimentation in software engineering", Springer Science and Business Media, 2012

- Xiao, L., Cai, Y., Kazman, R., Mo, R., and Feng, Q., “Identifying and quantifying architectural debt”, *38th International Conference on Software Engineering (ICSE)*, IEEE/ACM, pp. 488-498, Austin, TX, USA, May 2016.
- Yin R. K., “Case Study Research and Applications: Design and Methods”, Sage Publications, 3rd Edition, 2003.
- Yli-Huumo, J., Maglyas, A., and Smolander, K., “How do software development teams manage technical debt? –An empirical study”, *Journal of Systems and Software*, Elsevier, 120, pp. 195-218, 2016.
- Zazworka, N., Seaman, C., Shull, F., “Prioritizing design debt investment opportunities”, *2nd International Workshop on Managing Technical Debt (MTD’ 11)*, ACM, 2011.
- Zazworka, N., Shaw, M., Shull, F., and Seaman, C., “Investigating the impact of design debt on software quality”, *2nd Workshop on Managing Technical Debt (MTD ‘11)*, ACM, pp. 17-23, Hawaii, USA, 21 -28 May 2011.
- Zazworka, N., Vetró, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., and Shull, F., “Comparing four approaches for technical debt identification”, *Software Quality Journal*, Springer, 22 (3), pp. 403 – 426, Sept. 2014.
- Zhang, H. and Babar, M. A., “On Searching Relevant Studies in Software Engineering”, *14th International Conference on Evaluation and Assessment in Software Engineering (EASE’ 10)*, pp. 1-10, 12-13 April 2010, Keele, UK.
- Zhou, Y. and Leung, H., “Predicting Object-Oriented Software Maintainability using Multivariate Adaptive Regression Splines”, *Journal of Systems and Software*, Elsevier, 80 (8), pp. 1349-1361, 2007.

Index

- artifact, 49, 50, 78, 99, 103, 114, 121, 125
- benefit, II, IV, 13, 15, 22, 23, 26, 28, 29, 38, 39, 42, 43, 45, 49, 53, 54, 56, 90, 95, 96, 98, 113, 149, 150, 151, 152, 154, 155, 159, 161, 162, 164, 165, 167, 169, 170, 176, 177, 178, 179, 184, 185, 186, 187, 188, 194, 204, 205, 206, 212, 229, 240
- benefits, II, IV, 13, 14, 15, 18, 20, 23, 43, 44, 89, 90, 110, 114, 116, 149, 150, 151, 152, 154, 155, 158, 160, 161, 162, 164, 165, 167, 169, 172, 175, 176, 188, 190, 191, 193, 204, 210, 238
- case study, II, 15, 18, 19, 21, 43, 60, 62, 63, 65, 76, 77, 104, 105, 106, 109, 110, 114, 115, 116, 118, 126, 129, 134, 138, 145, 146, 147, 148, 149, 151, 154, 162, 166, 167, 168, 172, 177, 184, 187, 188, 193, 194, 196, 197, 198, 201, 202, 203, 204, 205, 233, 235, 238, 239, 241, 242
- code TD, III, 12, 18, 99, 105, 110, 113, 145, 147, 194
- communication, IV, 9, 12, 14, 22, 23, 24, 56, 59, 153, 180, 182, 184, 191
- cost, II, IV, 14, 22, 23, 27, 28, 38, 40, 41, 42, 43, 44, 45, 49, 53, 54, 56, 61, 83, 86, 87, 88, 92, 95, 96, 98, 102, 114, 115, 117, 119, 120, 131, 146, 149, 150, 151, 152, 154, 155, 156, 157, 159, 161, 162, 164, 165, 167, 169, 170, 173, 174, 175, 177, 178, 179, 180, 183, 184, 185, 186, 187, 188, 194, 196, 201, 202, 203, 204, 205, 206, 211, 212, 224, 229, 236, 238, 239, 240, 243
- cost/benefit analysis, 22, 23, 43, 45, 53, 54, 56, 204
- costs, II, III, 10, 11, 13, 14, 15, 18, 20, 28, 43, 44, 87, 90, 92, 97, 110, 117, 119, 145, 150, 152, 154, 155, 156, 157, 160, 161, 162, 164, 170, 173, 175, 179, 186, 190, 191, 193, 194, 201, 203, 204, 205, 212, 238, 239, 243
- debt, III, 9, 10, 13, 14, 16, 17, 18, 21, 22, 24, 25, 26, 27, 28, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 49, 50, 51, 52, 53, 54, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69, 71, 72, 73, 75, 76, 77, 78, 79, 85, 86, 87, 88, 90, 91, 92, 95, 96, 97, 102, 105, 112, 113, 114, 115, 116, 117, 118, 128, 145, 146, 147, 148, 154, 165, 193, 194, 196, 197, 198, 199, 200, 201, 202, 203, 204, 205, 206, 211, 233, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243, 244, 245, 246
- decision, 1, II, IV, 12, 14, 18, 20, 28, 45, 61, 91, 94, 96, 105, 123, 131, 147, 149, 150, 151, 152, 153, 154, 159, 160, 161, 162, 164, 165, 166, 167, 169, 170, 172, 173, 174, 175, 176, 177, 180, 181, 182, 183, 184, 185, 187, 188, 190, 191, 192, 193, 194, 195, 199, 200, 204, 205, 209, 210, 211, 230, 240, 241, 242, 244
- decision-making, II, IV, 14, 18, 97, 131, 149, 151, 152, 153, 154, 159, 166, 167, 169, 172, 175, 176, 177, 180, 181, 183, 184, 185, 186, 187, 188, 190, 191, 193, 194, 195, 205, 209, 210, 240, 242
- defect, 11, 42, 44, 46, 56, 115, 117, 119, 205, 241
- design, 9, 10, 11, 12, 13, 16, 23, 24, 37, 40, 42, 45, 46, 49, 50, 51, 56, 60, 61, 62, 63, 65, 67, 68, 71, 72, 73, 78, 84, 86, 89, 92, 96, 98, 99, 100, 101, 104, 109, 110, 113, 114, 117, 119, 122, 134, 143, 147, 151, 152, 153, 155, 160, 166, 169, 170, 171, 194, 198, 200, 201, 206, 233, 237, 238, 242, 244, 245
- design debt, 42, 72, 73, 201, 206, 245
- design quality, 72, 92, 98, 99, 233

documentation, 12, 24, 46, 51, 66, 67, 68,
71, 72, 88, 91, 98, 145, 149, 153, 154,
183, 185, 194, 202, 211, 241, 245

economics, II, III, 9, 13, 22, 23, 24, 31, 36, 39,
40, 41, 78, 79, 80, 81, 84, 85, 86, 88, 93,
112, 113, 143, 154, 184, 195, 235

effort, II, 9, 10, 11, 13, 14, 15, 23, 32, 38, 40,
41, 42, 44, 49, 50, 52, 54, 61, 67, 71, 78,
79, 84, 86, 87, 88, 89, 90, 91, 93, 95, 96,
97, 98, 99, 100, 102, 105, 106, 108, 114,
115, 117, 118, 119, 120, 121, 123, 124,
126, 127, 128, 146, 150, 151, 192, 193,
194, 197, 202, 203, 205, 211, 212

financial, II, III, 11, 13, 14, 16, 18, 19, 20, 21,
22, 23, 24, 25, 26, 27, 28, 29, 30, 32, 33,
34, 35, 37, 38, 39, 40, 41, 42, 43, 44, 45,
46, 49, 52, 53, 54, 56, 57, 58, 59, 78, 79,
85, 86, 117, 149, 150, 159, 162, 165, 169,
175, 180, 181, 184, 190, 191, 192, 193,
204, 211, 228, 233

identification, 11, 13, 24, 26, 31, 41, 54, 57,
101, 115, 143, 146, 156, 201, 246

interest, III, 9, 10, 11, 13, 14, 15, 17, 18, 21,
23, 26, 27, 28, 31, 37, 38, 39, 40, 41, 42,
43, 44, 45, 51, 53, 63, 66, 78, 79, 80, 81,
82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92,
93, 95, 96, 97, 98, 99, 100, 101, 102, 103,
104, 105, 107, 108, 109, 110, 111, 112,
113, 114, 115, 116, 117, 118, 119, 120,
123, 127, 128, 137, 138, 142, 143, 144,
146, 147, 148, 152, 166, 172, 191, 192,
193, 194, 199, 201, 202, 203, 206, 211,
212, 233, 237, 239, 242

interest probability, 43, 86, 202

investment, II, 14, 26, 27, 28, 29, 38, 44, 49,
50, 61, 81, 82, 83, 90, 149, 156, 159, 179,
191, 201, 205, 245

literature review, II, 13, 15, 16, 25, 29, 59, 79,
84, 101, 125, 157, 190, 192, 233, 238, 239

maintainability, III, 9, 10, 13, 14, 23, 40, 46,
50, 60, 61, 63, 64, 65, 66, 67, 69, 70, 71,
73, 74, 75, 76, 77, 100, 101, 103, 109,
110, 115, 116, 117, 118, 119, 120, 121,
122, 123, 124, 125, 127, 128, 130, 139,
140, 141, 146, 149, 160, 169, 192, 193,
212, 236, 240, 242, 243, 245

maintenance, II, III, 9, 11, 13, 17, 23, 41, 44,
46, 51, 61, 66, 67, 68, 71, 76, 78, 79, 86,
87, 88, 89, 90, 92, 95, 97, 99, 100, 102,
105, 106, 109, 110, 113, 115, 117, 118,
119, 120, 123, 124, 126, 128, 130, 143,
144, 146, 150, 155, 174, 191, 192, 193,
195, 201, 212, 235, 242

management, IV, 9, 13, 16, 17, 24, 27, 28, 29,
30, 32, 33, 37, 38, 39, 42, 43, 45, 46, 47,
52, 53, 56, 57, 59, 61, 62, 65, 71, 75, 77,
78, 86, 91, 95, 97, 150, 153, 154, 180,
197, 204, 205, 228, 233, 237, 240, 243

market, 9, 11, 13, 14, 15, 18, 21, 22, 23, 28,
29, 38, 44, 49, 50, 61, 67, 75, 81, 82, 83,
96, 149, 150, 159, 160, 161, 170, 186,
193, 195, 212

measurement, 11, 21, 24, 31, 39, 41, 42, 59,
79, 120, 146, 198

metrics, 11, 12, 13, 14, 41, 44, 54, 63, 99,
100, 101, 102, 103, 106, 112, 115, 116,
117, 118, 120, 121, 125, 130, 131, 133,
136, 139, 146, 212, 234, 243

monetization, 17, 170, 186

monitoring, 12, 19, 24, 30, 39, 42, 54, 56, 61,
63, 65, 79, 91, 95, 110, 113, 146, 199,
211, 237, 244

PAF, 150, 152, 155, 156

portfolio management, 22, 23, 29, 43, 45, 56,
205

principal, 9, 11, 13, 14, 17, 18, 23, 26, 27, 38,
40, 42, 43, 45, 49, 51, 64, 68, 71, 76, 78,
88, 89, 91, 92, 93, 95, 96, 97, 98, 102, 103,
104, 105, 108, 109, 110, 111, 112, 113,
114, 115, 116, 117, 118, 121, 123, 128,
137, 138, 139, 142, 143, 144, 147, 194,
201, 202

prioritization, 12, 14, 24, 44, 56, 61, 65, 66,
75, 76, 79, 112, 144, 146, 148, 152

quality, II, 9, 10, 11, 12, 13, 17, 22, 23, 24, 30,
32, 35, 41, 44, 45, 46, 49, 50, 51, 54, 56,

60, 61, 62, 64, 65, 66, 67, 68, 69, 73, 74,
75, 76, 77, 78, 86, 87, 89, 91, 92, 95, 98,
99, 100, 103, 106, 107, 110, 112, 114,
115, 116, 118, 119, 120, 121, 124, 125,
129, 130, 131, 133, 141, 143, 144, 145,
149, 150, 152, 154, 155, 156, 159, 169,
172, 176, 179, 188, 192, 193, 194, 196,
201, 202, 203, 205, 209, 210, 211, 212,
237, 238, 239, 240, 243, 245

quality attributes, 17, 24, 45, 60, 62, 64, 65,
66, 67, 68, 69, 73, 75, 76, 77, 100, 116,
143, 145, 149, 152, 154, 169, 192, 194,
211

real-options, 23

repayment, 11, 12, 15, 24, 26, 38, 43, 51, 53,
61, 62, 71, 76, 79, 85, 87, 89, 90, 91, 93,
97, 112, 144, 148, 193

risk, 26, 27, 28, 29, 38, 42, 45, 51, 86, 151,
153, 155, 180, 181, 184, 186, 195, 203,
204, 235, 243

software evolution, 43, 87, 91, 95, 97, 188,
235

SonarQube, 10, 42, 46, 98, 102, 109, 110,
114, 116, 121, 122, 130, 145, 203, 205

TD, II, III, IV, VI, 9, 10, 11, 12, 13, 14, 15, 17,
18, 19, 20, 21, 22, 24, 31, 33, 34, 35, 36,
37, 38, 39, 40, 41, 42, 49, 51, 53, 54, 58,
59, 60, 61, 62, 63, 64, 65, 66, 67, 68, 69,
71, 72, 73, 74, 75, 76, 77, 78, 79, 84, 85,
86, 87, 88, 89, 90, 91, 93, 95, 97, 98, 104,
105, 106, 107, 108, 109, 110, 112, 113,
114, 115, 116, 117, 118, 119, 120, 121,
122, 123, 124, 126, 127, 128, 130, 131,
132, 133, 134, 135, 136, 137, 138, 139,
140, 141, 142, 143, 144, 145, 146, 147,
148, 190, 191, 192, 193, 194, 195,
201, 202, 203, 204, 208, 211, 228

TD items, 12, 15, 42, 77, 97, 113, 115, 144,
204

TD management, 13, 14, 59, 76, 91, 112,
128, 148, 190

TD metaphor, 9, 14, 18, 112, 113, 193

TDM, II, IV, 11, 12, 14, 16, 17, 18, 24, 30, 33,
34, 39, 42, 45, 49, 54, 60, 62, 63, 76, 79,
86, 89, 95, 130, 144, 146, 190, 191, 192,
201, 203, 204

Technical Debt, 1, II, III, V, VI, 9, 11, 12, 13,
18, 20, 21, 22, 23, 24, 31, 34, 35, 36, 39,
40, 43, 46, 52, 60, 61, 62, 68, 70, 71, 72,
73, 78, 79, 84, 86, 87, 93, 95, 96, 111, 112,
119, 146, 192, 196, 197, 198, 199, 200,
201, 202, 203, 204, 205, 233, 234, 235,
236, 237, 239, 240, 241, 242, 243, 244,
245

Technical Debt Management, 1, II, V, 9, 11,
24, 60, 61, 79, 95, 192

trade-offs, 9, 22, 23, 63, 65, 69, 76, 180, 192,
228

value, 22, 23, 27, 28, 33, 37, 38, 39, 42, 44,
49, 50, 52, 53, 74, 79, 99, 100, 101, 103,
107, 111, 117, 124, 125, 129, 134, 137,
139, 140, 141, 146, 151, 164, 177, 197,
203, 204, 206