Universidad Politécnica de Madrid

Escuela Técnica Superior de Ingenieros Informáticos

Modelo de Estimación de Valor para la Toma de Decisiones Relacionadas con la Evolución de Productos Software.

Tesis doctoral

Ing. Carlos Fernández Sánchez

2017

Universidad Politécnica de Madrid

Departamento de Lenguajes y Sistemas Informáticos e Ingeniería de Software

Escuela Técnica Superior de Ingenieros Informáticos

Modelo de Estimación de Valor para la Toma de Decisiones Relacionadas con la Evolución de Productos Software.

Ing. Carlos Fernández Sánchez

Supervised by

Dr. Juan Garbajosa

2017

This thesis has been partially supported by: (i) UPM (Technical University of Madrid) under their Researcher Training program. (ii) The Spanish Ministry of Science and Innovation (MICINN) through the R&D+i projects INNOSEP (TIN2009-13849) and i-Smart Software Factory (IPT-430000-2010-038). (iii) The Spanish Ministry of Industry (MITYC) through the R&D project IMPONET (ITEA 2 09030 — TSI-02400-2010-103). (iv) The Centre for Industrial Technological Development (CDTI) through R&D projects NEMO&CODED (ITEA2 08022 — IDI-20110864) and Energos (CEN-20091048). (v) The Spanish Ministry of Economy and Industry (MINECO) through the R&D+i projects MESC (DPI2013-47450-C2-2-R) and CrowdSaving (TIN2016-79726-C2-1-R).

Abstract

Most of the software is produced as a result of a continual subsequent evolution. Decisions on software evolution are focused on what new features to add, improve, or even remove. Additionally, there are decisions about what part of the software should be improved to increase the capacity of evolution, what bug have to be solved, or what new technologies should be included in the product. But to make informed decisions about software evolution it is necessary to know the value of implementing such decisions. Traditionally, value-based decisions have been made based on the visible characteristics of software (external quality). But many decisions can be focused on improving the capacity of the software product to be changed in the future without adding visible characteristics (internal quality).

This thesis defines a customer value model that help make decisions in a context of continuous evolution of software. To do that, this thesis provides four main contributions: (i) the identification of how changes focused on improving the internal quality of software add value for the customer; (ii) the definition of a theoretical framework based on the elements that are required to define models for managing the evolution of software products; (iii) the identification of the available methods and strategies to be used within the model for managing the evolution of software products; (iv) a model to be used in software projects to manage their evolution considering the internal and external quality of software.

The model was used in a case study based on the analysis of a large software project.

Keywords: value-based software engineering, making decisions, technical debt, technical debt management, estimation model.

Resumen

La mayoría del software se produce como resultado de una serie de continuas evoluciones. Las decisiones en la evolución del software se centran en qué nuevas características añadir, mejorar o incluso eliminar. Además hay decisiones sobre qué partes del software mejorar para incrementar la capacidad de evolución del mismo, qué errores corregir o qué nuevas tecnologías incluir en el producto. Pero para tomar buenas decisiones sobre la evolución del software es necesario saber el valor de implementar dichas decisiones. Tradicionalmente, las decisiones basadas en valor se han hecho teniendo en cuenta las características visibles del software (calidad externa). Pero muchas decisiones pueden estar centradas en mejorar la capacidad del producto software para ser cambiado en el futuro sin añadir características visibles (calidad interna).

Esta tesis define un modelo de valor para el cliente que ayuda a la toma de decisiones en un contexto de evolución continua del software. Para lograr esto, esta tesis provee cuatro contribuciones principales: (i) la identificación de cómo los cambios que tienen como objetivo mejorar la calidad interna del software añaden valor para el cliente; (ii) la definición de un marco de trabajo teórico basado en los elementos que son necesarios para definir modelos para la gestión del la evolución de productos software; (iii) la identificación de los métodos y estrategias disponibles para ser usados junto con el modelo de evolución de productos software; (iv) un modelo para ser usado en proyectos software para gestionar su evolución considerando la calidad interna y externa del software.

El modelo ha sido usado en un caso de estudio en el que se ha analizado un proyecto software grande.

Palabras clave: ingeniería del software basada en valor, toma de decisiones, deuda técnica, gestión de deuda técnica, modelo de estimación.

A Mabel, mi familia y mis compañeros en la Universidad Politécnica de Madrid. Ellos han hecho posible este trabajo.
To Mabel, my family, and my colleagues at Universidad Politécnica de Madrid. They made this work possible.

"Adde parvum parvo magnus acervus erit."
"Add little to little and there will be a big pile."
Publio Ovidio Nasón

Acknowledgements

Me gustaría dar las gracias a Mabel por su constante apoyo. Gracias a ella el trabajo siempre fue menos duro.

Igualmente quiero dar las gracias a mis padres, Ovidio y Emilia. Ellos son los que me han inculcado los valores y educación sin los que ni tan siquiera podría haber comenzado a estudiar.

También quiero recordar al resto de mi familia, en especial a mis hermanas por su apoyo y a mis abuelos por mostrarme otras formas de ver el mundo.

Quiero especialmente dar las gracias a mi director de tesis Juan Garbajosa por su orientación y paciencia. Gracias a sus enseñanzas yo he podido hacer esta tesis, que de otro modo, nunca hubiese completado. Igualmente muchas gracias al resto de miembros del grupo SYST, los actuales y los que han pasado por él en los últimos años. Ha sido todo un placer trabajar con vosotros durante el tiempo que me ha llevado completar esta tesis y espero seguir trabajando con vosotros muchos años más. Sobre todo gracias a Agustín, Jennifer, y Jessica con quienes he trabajado codo con codo en muchos proyectos y que me han ayudado cuando lo he necesitado. Igualmente gracias a Carlos S., Carlos V., Dani e Iria aunque vuestro camino os ha llevado lejos de SYST, fue un placer trabajar con vosotros. No quiero olvidarme de los nuevos compañeros con los que espero seguir trabajando después de esta tesis: Javier, Jorge, Norberto y Héctor. En general gracias a todos los compañeros, amigos, alumnos y profesores que de una u otra forma han contribuido a que yo pudiese realizar esta tesis.

Thank you to Prof. Pekka Abrahamsson and Prof. Xiaofeng Wang for their support, helpful advice and comments during my stay at the Free University of Bozen-Bolzano.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

1.1 Research Motivation

In 2002, Lehman and Ramil [67] already pointed out that most of the software is produced as a result of a continual subsequent evolution. Continuous software engineering [41, 15] somehow has extended and formalized this way of understanding software development. To express the concept of evolution, Schmid [98] used the concepts of external and internal quality. Following Schmid [98], external quality of a software product is the accumulation of any observable quality of the product at run time (e.g., for example, a functionality, and a level of performance); internal quality of a software product is the accumulation of any non-observable quality of the product at run time (e.g., code quality, complexity of the code, and number of source files). Schmid [98] stated that whereas an evolution step can describe any form of change in the external quality of the product, internal quality changes by themselves are not evolution steps (e.g., refactoring). Following also Schmid [98], within this thesis, the evolution of a product is understood as any form of change that leads to an observable effect at a system level. This thesis is focused on the decisions that have to be made in the context of this evolution.

Decisions on software evolution are focused on what new features to add, improve, or even remove. Additionally, there are decisions about what part of the software should be improved to increase the capacity of evolution, what bugs have to be solved, or what new technologies should be included in the product. But to make informed decisions about software evolution in software engineering it is necessary to know the value of implementing such decisions. The term value can be used with many different meanings. In this thesis, the term value is used to refer to customer value. Any other acceptation of value will be explicitly indicated. In software products, value is mainly added to the products by increasing their external quality. External quality is the totality of characteristics of the software product from an external view. [1]. That is, the external quality is the quality that is perceived by the users, it is mainly added by the visible characteristics of the products, and consequently, it is related to the customer value. But some changes can improve the capacity of the software to be changed. That is, that kind of changes will make easier to increase the external quality of the software in the future. This second type of changes increases the internal quality of the software. Internal quality is the totality of characteristics of the software product from an internal view [1]. The internal quality is the quality that indicates how well is constructed a product internally and therefore, it includes the capacity of such product to be changed and evolved. Thus, decisions about software evolution have to consider both internal and external quality of the software. Hence, decisions in software evolution will be between adding external quality by changes that the customer can perceive (e.g., new features) and improve the internal quality (e.g., refactoring to add flexibility) to facilitate future changes, and consequently to make easier to increase the external quality in the future.

Changes in the evolution of software typically consume about 40 to 80 percent (60 percent average) of software costs [44]. Of that effort, much is due to adding new capability [44]. In many cases, software products are evolved over years to adapt their functionality to the changes in their context. Furthermore, a program that is used in a real-world environment necessarily must change or become progressively less useful in that environment [66]. In practice, software usually follows a continuous evolution to deliver new characteristics [41]. Therefore, decisions about the evolution of software products have a big impact on the economic performance and success of software companies.

In the real world, organizations have limited resources to evolve software and therefore they have to select what evolution steps are the most profitable. But make these decisions about software evolution is not an easy task. Many factors have to be taken into account. In a competitive market, not only the quality added, internal or external, is relevant, there is other important factors: time-to-market, competitive products, type of market where the product will be launch, etc. It is difficult to make decisions without a holistic view of software development, including business strategy [41].

Software release planning, or road-mapping, has as goal to select an optimum set of features or requirements to deliver in a release within given constraints [110]. There are several literature reviews on software release planning models [95][110][7]. These studies analyzed a total of 45 models for software release planning. For the goal of the present thesis, it is possible to remark some main findings in those studies. There is a many models to prioritize requirements and release planning and therefore there is a large work already done in the prioritization and selection of characteristics for software evolution focused on the external quality of software [95][110][7]. But on the contrary, software release planning scientific proposals have not yet reached the maturity required by industrial contexts [7]. Additionally, these methods do not deal with possible changes in internal quality together with the implementation of new features. Actually, there is not a major focus on addressing system constraints [95]. And when the models deal with technical constraints these constraints are focused on the requirements themselves and the ability to implement them (e.g., dependencies between requirements) [110]. The models do not deal with possible constraints due to internal quality issues.

A software has the capacity of being changed, and depending on different internal attributes, this software will be more or less easy to be changed. Several literature reviews have analyzed the software capacity for evolution [12], prediction techniques for software evolution [92], and software architecture evolution [17]. Software evolution research area is by its nature, due to its complexity, more difficult to be explained by theoretical principles than by practical experiences; thus, a theoretical foundation with practical value for software evolution is necessary [17]. Additionally when software evolves some of the initial design decisions that were made during the system's creation may no longer hold, and the architecture may not facilitate certain changes [41]. In this situations, shortcuts can be made and software decay materializes. Therefore, software evolution drives to increase the cost of change of software if one does not actively work against this.

Traditionally, value-based decisions have been made based on external quality, that is, the value added to the customers due to new or improved functionality. But many decisions can be focused on improving the capacity of the software product to be changed in the future without adding external quality. That is, decisions have to consider not only the value added in the present but the possible value that could be added in the future. Many models deal with the evolution of software from the point of view of external quality [95, 110, 7], whereas techniques and methods that are focused on internal quality usually do not consider the business point of view [39]. Therefore, there is a lack in models that consider both points of view.

Therefore, in making decisions about software evolution, it is necessary to consider both internal and external quality of software, business strategy, and short and long-term customer value. The achievement of this goal implies to know the capacity for changes of software and to identify when it is more profitable to invest in increasing such capacity or when it is better to add more external quality to increase in a direct way the value for the customer of the product.

1.2 Research Context

This thesis is focused on studying software evolution from a value-based perspective. Therefore the thesis is framed in the next software engineering areas:

- **Value-Based Software Engineering.** Value-Based Software Engineering is a paradigm that uses value as a driver for developing software. We use the definition provides by Biffl et al's book *Value-Based Software Engineering* [13]: "The value perspective provides a good way of looking at the product development process. The ultimate aim of value propositions in software engineering is to create a strategy to achieve long-term profitable growth and sustainable competitive advantage for software companies. The implication is that software developers need to consider the key elements of value in terms of how to create value for current as well as future software products and how to deliver this value to a customer in the most profitable way. In other words, software developers should have a better understanding of the implications of the decisions they have made about the software product, the software development process, and the resources that they use."

- **Continuous Software Engineering.** The ultimate goal of Continuous Software Engineering is to take a holistic view of a software production entity, whether this is a single software organization or an ecosystem where different organizations together deliver a final product [41]. The link between business strategy and software development ought to be continuously assessed and improved [41]. To achieve this goal, Fitzgerald and Stol [41] defined several continuous activities that should be done in software development. This thesis is mainly focused on the following activities of Continuous Software Engineering:
  - **Continuous planning.** Holistic endeavor involving multiple stakeholders from business and software functions whereby plans are dynamic open-ended artifacts that evolve in response to changes in the business environment and thus involve a tighter integration between planning and execution.
  - **Continuous evolution.** Most software systems evolve during their lifetime. However, a system's architecture is based on a set of initial design decisions that were made during the system's creation. Some of the assumptions underpinning these decisions may no longer hold, and the architecture may not facilitate certain changes. In the last years, there has been increased focus on this topic. When an architecture is unsuitable to facilitate new requirements but shortcuts are made nevertheless, technical debt is incurred.
  - **Continuous delivery.** Continuous delivery is the practice of continuously deploying good software builds automatically to some environment, but not necessarily to actual users.
- **Technical Debt Management.** The term technical debt is a metaphor that refers to the consequences of weak software development [28]. Technical debt can grow because of the inability of developers to develop high-quality applica-

tions [114] or as the result of decisions that prioritize functionality over quality. Moreover, technical debt can evolve because of circumstances that are beyond the developers' control: for example, it can arise because of changes in the context of the application [19], and it is cleared automatically at the end of a system's life [20]. Some technical debt is inevitable in a world with finite resources [114]: it is both, possible and necessary to live with some technical debt. However, several studies have highlighted the negative effects of uncontrolled technical debt on software development [87], including on the developer's morale, team velocity, and the quality of the product [114][83]. The problem is that the benefits of refactoring software to remove technical debt are largely invisible, sometimes intangible, and usually, long term, whereas the costs of refactoring activities are significant and immediate [21]. Technical debt management consists of identifying the sources of the extra costs of software maintenance and determining whether it is profitable to invest efforts into improving a software system [114]. If technical debt is not managed could become too high to sustain. The company then will be compelled to invest all its efforts into keeping the system running, rather than increasing the value of the system by adding new capabilities [114], which will seriously damage the company's profitability and risk the inability to fulfill its strategic goals. In any case, from *a business point of view*, reducing technical debt is a good idea if, and only if, it leads to increased profitability [113]. Thus, decision-making is essential in technical debt management.

1.3 Research Question

This thesis aims to respond the following research question:

RQ 1 *Could one have a model that considers customer value together with the short and long-term impact of decisions to help support the decision-making process in software evolution? What elements would this model be made of?*

1.4 Research Contributions

The main contribution of this thesis can be formulated as follows:

> "A model oriented to the customer value that helps make decisions in a context of continuous evolution of software considering internal characteristics of software and the short and long-term impact of such decisions."

This main contribution can be refined in the following more specific contributions and key publications resulting from this thesis. Publications are described in Section 1.4.1.

Contribution 1 *Identification of how software internal quality increases the customer value.*

It was found that the technical debt concept, through principal, interest, and interest probability, helps to reason about the value of changes in the internal quality of software. Technical debt links improvements in the internal quality of software with the value that could be added to the software in the future. Therefore, technical debt should be considered for decision making with a value-based software engineering perspective.

Publications: P1, MT1.

Contribution 2 *Identification and definition of the elements that are required to create models that help make decisions in software evolution.*

At that point, this thesis was focused on technical debt. Therefore, the elements of technical debt management were identified. Based on the elements analysis it was possible to define a framework. That framework could be used to define models for technical debt management for specific systems with two objectives: to demonstrate how the elements work in practice and to implement specific technical debt management models based on the integration of the tools that are currently available for the management of technical debt.

In addition to the framework, an important finding of this thesis is that any decision about software evolution implies a trade-off between software release characteristics and technical debt removal considering the business constraints.

It is also important to highlight that while time-to-market was one of the least suggested elements in technical debt management literature, it was probably the most referenced cause of technical debt. According to [114], it was one of its most relevant antecedents. This is an interesting paradox and a serious issue because managing technical debt without considering time-to-market could lead to wrong decisions that could affect important deadlines in a project.

Publications: P5, P7.

Contribution 3 *Identification of tools and strategies that support the elements identified in Contribution 2 and the lacks in this support.*

There is not enough support for all the technical debt management elements. Essential elements are not currently covered, such as time-to-market. Other elements also require more support, especially from the business organizational management point of view. Different strategies for technical debt management are focused on different elements. And therefore, there are not tools or strategies that support all the elements. As a consequence, it was necessary to go further in the integration of tools and strategies to manage effectively technical debt. Consequently, TEDMA Tool has been implemented. TEDMA is designed to integrate third party tools and techniques. In fact, TEDMA facilitates the usage of the technical debt management elements in real projects.

Publications: P4, P7, P9, T1.

Contribution 4 *Definition of a model for making decisions on software evolution using the elements identified in Contribution 2 and that integrates tools and strategies identified in Contribution 3.*

This contribution consists of a model for technical debt management that considers time-to-market. The model uses the technical debt management framework produced in Contribution 2 as a guide, which was useful in adapting the time-to-market modeling framework, which came from a different domain to technical debt management. This model is a step forward in making decisions in software engineering, especially in the case of trade-offs of the internal and external quality of software when considering an economic point of view. It was possible to deploy the model for the technical debt management in a case study using a large project. For the case study execution, it was necessary to use the techniques and strategies identified by Contribution 3 as well as TEDMA Tool. The case study's execution demonstrates that it is possible to use the selected time-to-market modeling framework for technical debt management and that it is necessary a holistic perspective in technical debt management that includes the business perspective, that is, economic constraints and business goals.

Publications: P2, P3, P8, P9, T1.

1.4.1 Publications

The achievement of the contributions of this thesis has resulted in several publications and other outcomes (a master thesis and a tool).

They are summarized in Table 1.1 and Table 1.2.

MT 1 *Carlos Fernández-Sánchez, Value Considerations in Software Architecture: A Systematic Literature Review, Master thesis degree on Máster Universitario en Software y Sistemas, Universidad Politécnica de Madrid, July 2012.*

MT1 focused on how value can be added from the internal quality of software. To do that, it analyzed the relationship between value and software architecture. One of its main findings is the possibility of use technical debt management to manage how value can be added in the software evolution.

P 1 *Carlos Fernández, Daniel López, Agustín Yagüe, and Juan Garbajosa. Towards estimating the value of an idea, 2011 Workshop on Managing the Client Value Creation Process in Agile Projects (VALOIR) in conjunction with the 12th International Conference on Product Focused Software Development and Process Improvement (Profes '11). ACM, New York, NY, USA, 62-67.*

P1 [34] focused on software innovation and how to deal with the value estimation. This publication was one of the results of the state of the art study about value in software engineering. It was a preliminary study that pointed out the identification of sources of innovation in software development and a short list of elements that could be needed to estimate the value of "new ideas" in software development.

P 2 *Fernández Sánchez, Carlos; Díaz Fernández, Jessica; Garbajosa Sopeña, Juan y Pérez Benedí, Jennifer (2013). A Cost-Benefit analysis model for technical debt management considering uncertainty and time. Work in progress track at the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2013) Santander, Spain, September 4-6, 2013.*

P2 [38] sketched some basic concepts that are necessary for cost-benefit analysis based on technical debt by presenting a new model that allows performing cost-benefit analysis dealing with the uncertainty of the future changes and considering the time frame where decisions can be made. In addition, the model helps to reason about the possible situations that can occur around the system and that can affect the software evolution.

P 3 *C. Fernández-Sánchez, J. Díaz, J. Pérez and J. Garbajosa, Guiding Flexibility Investment in Agile Architecting. In proceeding of the 47th Hawaii International Conference on System Sciences, Waikoloa, HI, 2014, pp. 4807-4816, BEST PAPER NOMINATION.*

P3 [35] used the model defined in P2 in a case study performed in the development of a software product. The model was integrated into a process within the software development methodology and used to make decisions about investments in flexibility (refactorings) in the architecture. The model was satisfactorily used allowing several findings that guided the next research steps. The model needed to be adapted to the context of the software under analysis, the case study was a framed project about a very specific decision, and therefore, to use the model for more general decisions in any possible project more adaptations would be required. As a conclusion, it was necessary to study what elements were required to create models in other projects.

P 4 *C. Fernández-Sánchez, J. Garbajosa, C. Vidal and A. Yagüe, An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture Perspective, 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, in conjunction with the 37th International Conference on Software Engineering, Florence, 2015, pp. 22-28.*

P4 [36] showed the methods and techniques that were available to support the identified elements in the previous study. This allows identifying gaps in the state of the art and available techniques to be used in the implementation of specific models to making decisions about software evolution.

P 5 *C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, A framework to aid in decision making for technical debt management, IEEE 7th International Workshop on Managing Technical Debt (MTD) in conjunction with 31st International Conference on Software Maintenance and Evolution, Bremen, 2015, pp. 69-76.*

P5 [37] identified the elements that had to be considered for creating models to make decisions about software evolution. This publication presents the first version of a framework with the goal of guiding in the creation of specific models for specific projects. It also includes different points of view depending on the stakeholders involved in the decision-making process.

P 6 *Software Startups – A Research Agenda (Michael Unterkalmsteiner, Pekka Abrahamsson, XiaoFeng Wang, Anh Nguyen-Duc, Syed Shah, Sohaib Shahid Bajwa, Guido H. Baltes, Kieran Conboy, Eoin Cullina, Denis Dennehy, Henry Edison, Carlos Fernández-Sánchez, Juan Garbajosa, Tony Gorschek, Eriks Klotins, Laura Hokkanen, Fabio Kon, Ilaria Lunesu, Michele Marchesi, Lorraine Morgan, Markku Oivo, Christoph Selig, Pertti Seppänen, Roger Sweetman, Pasi Tyrvinen, Christina Ungerer, Agustín Yagüe), In e-Informatica Software Engineering Journal, volume 10, 2016.*

P6 [116] is a collaboration with other researchers in the field of software startups. This publication corresponds to one of the exploitation strategies that are planned for this thesis. One of the fields where the outcomes of this thesis can be used is in making decisions in software startups. This publication is mainly related to future work.

Table 1.1: Summary of research objectives, results, and publications

Contribution	Result	Outcomes
Contribution1	Identification of technical debt management as a mechanism to manage value	P1, MT1
Contribution2	A theoretical framework for technical debt management	P5, P7
Contribution3	State of the art in methods and techniques for technical debt management	P4, P7, P9, T1
Contribution4	A model for technical debt management	P2, P3, P8, P9, T1

P 7 *Carlos Fernández-Sánchez, Juan Garbajosa, Agustín Yagüe, Jennifer Pérez, Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study, Journal of Systems and Software, Volume 124, February 2017, Pages 22-38, ISSN 0164-1212.*

P7 [39] is an extension of the P5 and P4 publications. This publication included a revision of the framework defined in P5 and extended the analysis of the techniques and methods of P4 by analyzing their rigor and industrial impact.

P 8 *Carlos Fernández-Sánchez, Juan Garbajosa, Jessica Díaz, Jennifer Pérez, The Relationship between Technical Debt Management and Time-to-Market: An Exploratory Case Study, Submitted to IEEE Transaction on Software Engineering.*

P8 (submitted) presents a model for making decisions on software evolution considering internal and external quality. The model was defined following the framework defined in P7. Finally, the model was applied to a large project in a case study.

P 9 *Carlos Fernández-Sánchez, Juan Garbajosa, Héctor Humanes, Jessica Díaz, An Open Tool for Assisting in Technical Debt Management, accepted in Euromicro DS-D/SEAA 2017.*

P9 (accepted) presents TEDMA Tool, the tool that has been developed in this thesis to use the model defined in P8.

T 1 *Carlos Fernández Sánchez, TEDMA Tool, A tool developed for technical debt management.*

1.5 Research Methodology

In this section, the methodology followed in this thesis is described.

Because the research started from an initial stage where no previous research was performed, an exploratory sequential mixed approach was used following the Creswell's

Table 1.2: Summary of the impact of the publications

Publication	Journal/Conference/Workshop	Impact
P1 [34]	Workshop on Managing the Client Value Creation Process in Agile Projects (VALOIR)	Main conference CORE B
P2 [38]	Work in progress track at the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)	Main conference impact CORE B
P3 [35]	Hawaii International Conference on System Sciences	**CORE A**
P5 [37]	IEEE International Workshop on Managing Technical Debt (MTD) in conjunction with International Conference on Software Maintenance and Evolution (ICSME)	Main conference CORE A
P4 [36]	IEEE/ACM International Workshop on Software Architecture and Metrics, in conjunction with the International Conference on Software Engineering (ICSE)	Main conference CORE A*
P6 [116]	e-Informatica Software Engineering Journal	CiteScore 2015: 0.30, SJR 2015: 0.117, SNIP 2015: 0.318
P7 [39]	Journal of Systems and Software	**JCR 2016: 1.444 (Q1)**, CiteScore 2016: 3.10, SJR 2015: 0.897, SNIP 2015: 2.415
P8 (submitted)	IEEE Transactions on Software Engineering	**JCR 2016: 3.272 (Q1)**, CiteScore 2016: 5.51, SJR 2015: 1.543, SNIP 2015: 4.423
P9 (accepted)	Euromicro Conference on Software Engineering and Advanced Applications (SEAA)	**CORE B**

recommendations [26, p. 266-274]. The mixed-methods approach of this research as outlined by Creswell [26] requires the use of both qualitative and quantitative data. In this thesis, qualitative data were extracted mainly from literature reviews and exploratory case studies. Qualitative data were firstly used to define a theoretical framework of reference and to identify quantitative measures and techniques to be used in later stages of the research. Quantitative data were obtained from software tools used to monitor the development process, the product status, and the code quality. Quantitative data were used for applying the proposed model in a large software project.

As a consequence, this thesis used several methods to perform different steps in the research. Each method was selected to achieve specific outcomes in the different steps of the thesis. In Section 1.5.1 the methodologies and methods used are described. In Section 1.5.2 the process followed, including each step and the justification of the methodology used in each step is provided.

1.5.1 Research Methodologies and Methods

Systematic Literature Review. A systematic literature review is a means of identifying, evaluating and interpreting all available studies relevant to a particular research question, or topic area, or phenomenon of interest [63]. Its importance is that it synthesizes existing work in a manner that is fair and seen to be fair [63]. In this thesis, the guideline provided by Kitchenham and Charters [63] for performing systematic literature review in software engineering was used. According to that guideline, the main characteristics of systematic literature review studies are:

- The motivation for performing a systematic literature review is to summarize the existing evidence concerning a treatment or technology.
- It helps to identify any gaps in current research in order to suggest areas for further investigation.
- It can be used to provide a framework/background in order to appropriately position new research activities.
- It uses empirical evidence to determine whether the given hypothesis is supported or contradicted.

Narrative Synthesis. As defined by Popay et al. [86] narrative synthesis refers to an approach, which can be used within systematic literature reviews, for synthesis of multiple studies and that relies primarily on the use of words and text to summarize and explain the findings of the studies. Whilst narrative synthesis can involve the manipulation of statistical data, the defining characteristic is that it adopts a textual approach to the process of synthesis to ‘tell the story’ of the findings from the included studies. As used here ‘narrative synthesis’ refers to a process of synthesis that can be used in systematic reviews focusing on a wide range of questions, not only those relating to the effectiveness of a particular intervention. In this thesis, a narrative synthesis was used, following the guideline of Popay et al. [86] for using narrative synthesis in systematic literature reviews.

Systematic Mapping. A systematic mapping study allows the evidence in a domain to be plotted at a high level of granularity [63]. This allows for the identification of evidence clusters and evidence deserts to direct the focus of future systematic reviews and to identify areas for more primary studies to be conducted [63]. A systematic mapping study provides a structure of the type of research reports and results that have been published by categorizing them and often gives a visual summary, the map, of its results [84]. Systematic mapping studies are designed to provide a broad overview of a research area [84]. A systematic map helps to identify research gap in a topic area and indications for lack of evaluation or validation research in certain areas with less effort than a systematic literature review [84]. In this thesis, systematic mapping was performed by following Petersen et al.'s guide [84].

Constant Comparison (Qualitative Research). One of the differences between systematic literature reviews and systematic mappings is that the latter do not require deep reading and synthesis of the analyzed studies [84]. In the present thesis, the studies selected for the systematic mapping were analyzed in detail to extract additional information. Specifically, the selected studies were analyzed using coding and continuous comparison, both techniques used in qualitative research [108]. These techniques were useful to identify common concepts in the studies and to create definitions of them by gathering all the information provided by the different studies. This thesis did not use Grounded Theory [108], but it used some of the techniques used in such methodology.

Rigor and Industrial Relevance. Systematic mapping studies are designed to provide a broad overview of a research area [84]. In addition to that, in this thesis, the selected studies in systematic mapping were analyzed in terms of rigor and industrial relevance. The method defined by Ivarsson and Gorschek [56] was used. This provided more detailed view of the state of the art of the research area from an industrial perspective.

Exploratory Case Study. A case study is an empirical method that investigates contemporary phenomena in its natural context [123]. Exploratory case studies are performed for finding out what is happening, seeking new insights and generating ideas and hypotheses for new research [94]. In this thesis, the Runesson and Höst's [94] guideline for conducting case studies in software engineering was used.

1.5.2 Research Process

Figure 1.1 shows the different steps and methods used in this thesis. This thesis is mainly an exploratory effort. Therefore, the research started studying the state of the art of wide scope concepts as software architecture and value (see the outcome of the first process in Figure 1.1). After some initial finding were found out, a second state of the art was performed in a more specific topic: technical debt management (see the outcome of the fourth process in Figure 1.1). Consequently, within this thesis, two states of the art were analyzed in different stages of the research. In the next paragraphs, each step is described.

- **Initial Exploratory Study.** The first phase consist of an exploration about how to use the concept of value in software

  The first step performed in this thesis was a systematic literature review [63] of value added from decisions in the internal quality of the software. To do that the systematic literature review was focused on software architecture. Software architecture is defined by ISO/IEC/IEEE 42010:2011 [2] as system fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution. The architecture of a system constitutes what is essential about that system, considering its relationship with its environment. Therefore, software architectures provide high-level abstractions for representing the structure, behavior, and key properties of complex software systems [42]. Then, using architecture, this thesis uses a top-down strategy. This strategy was motivated because there are too many low-level abstractions in software to be analyzed in terms of internal quality of software and value in an affordable way. Design, code practices, testing, documentation, etc. provide many different points of view of the internal quality of software. The fact that this thesis analyzes value from software architecture does not mean that these other abstractions levels were not used in other steps. But the initial step was to analyze how the internal quality of software is related to the value of software and therefore it is better to start analyzing the part of software more related with business, that in this case, it is software architecture [11, ch. 1].

Narrative Synthesis was used to identify the concepts involved in the value creation in software internal quality in the studies analyzed in the systematic literature review of the previous process. The motivation to use this method was due to the studies found in the systematic literature review provide very different points of view about how value can be created from the software architecture and this method is suitable to analyze heterogeneous conclusions from different studies.

One of the main identified concepts that can be used to analyze the value of internal quality decisions, is technical debt. With the outputs of the systematic literature review and the synthesis process, a preliminary case study was performed to analyze the applicability of the learned concepts for creating a model for technical debt management. The main goal of this process was to increase the knowledge by using available techniques on the field. The main outcomes of this process were the identification of a set of limits that exists for defining models for technical debt management, especially the necessity of adapting models for the context of each project.

- **Framework for Technical Debt Management.** Thus it was necessary to go further in the state in the art of technical debt management. To do that a systematic mapping study on technical debt management was performed to analyze the state of the art in this topic. A systematic mapping study helps to identify research gap in a topic area and indications for lack of evaluation or validation research in certain areas [84]. Therefore it was suitable for the goals of this step. But to go further than the state of the art on technical debt management this thesis use coding and constant comparison from qualitative research [108] to define a framework based on the identification and definition of the elements that are required for technical debt management. This was a synthesis step were the studies selected for the systematic mapping study were analyzed to identify and define the elements required for technical debt management. Thanks to the identification of the elements a theoretical framework about technical debt management was defined.

- **Method and Techniques for Technical Debt Management.** To know the support of methods and techniques for technical debt management we analyzed the rigor and industrial relevance of the studies identified in the systematic map-

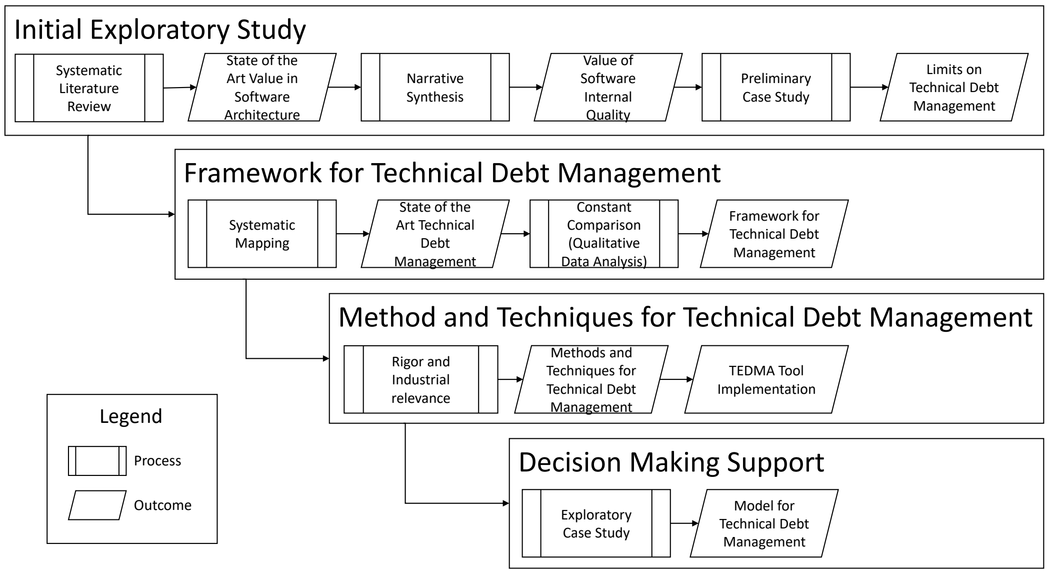

Initial Exploratory Study
Systematic Literature Review
State of the Art Value in Software Architecture
Narrative Synthesis
Value of Software Internal Quality
Preliminary Case Study
Limits on Technical Debt Management
Framework for Technical Debt Management
Systematic Mapping
State of the Art Technical Debt Management
Constant Comparison (Qualitative Data Analysis)
Framework for Technical Debt Management
Method and Techniques for Technical Debt Management
Rigor and Industrial relevance
Methods and Techniques for Technical Debt Management
TEDMA Tool Implementation
Decision Making Support
Exploratory Case Study
Model for Technical Debt Management
Legend
Process
Outcome

Figure 1.1: Research process followed in this thesis

ping study. This allowed analyzing the currently available techniques that are ready to be used in the industry. This provides a deeper understanding of the current state of the art in technical debt management and to select a set of techniques to be used in the case study that was performed in the next step and implemented in the tool that has been implemented within this thesis to integrate available tools and strategies for technical debt management. The development of the tool was guided by the outcomes of the analysis of the available techniques and methods performed in the previous process.

- **Decision Making Support.**

  Finally, using the framework for technical debt management and the available methods and techniques this thesis includes an exploratory case study [94] where a specific model for technical debt management was defined from the theoretical framework to analyze a large software project. To conduct the case study, the previously implemented software tool, TEDMA Tool, was used to extract and analyze the data of the project under analysis.

1.6 Thesis Overview

The remainder of this thesis is structured using the objectives described in Section 1.3. Therefore, the structure used is:

- Chapter 2 identifies how customer value is added to software from the internal quality of software. This chapter includes the state of the art of the usage of the value concept in software architecture.
- Chapter 3 presents a preliminary case study that was performed with the objective of obtaining further knowledge about how the concepts identified in Chapter 2 can be effectively used in software projects.
- Chapter 4 identifies the entire set of elements that have been used to manage technical debt. Elements are understood as the concepts used to implement technical debt management, regardless of their nature. This chapter includes the state of the art of technical debt management.
- Chapter 5 identifies and analyzes the tools and strategies for technical debt management that are available to support the elements identified in Chapter 4.
- Chapter 6 presents TEDMA, a tool for technical debt management. TEDMA implementation responds to the necessity of a tool that facilitates the experimentation with different technical debt management tools and strategies.
- Chapter 7 puts into practice the technical debt management element framework defined in Chapter 4. To do that a model for technical debt management that takes into account time-to-market has been defined and used in a case study.
- Chapter 8 presents and analyses the main contributions of this thesis and it also presents future research.

Part II

Identification of how software internal quality increases the customer value

Chapter 2

How to Add Value from Software Architecture

The goal of this chapter is to identify how value for the customer is added to software from the internal quality of software. This chapter is an excerpt from the following work:

Carlos Fernández-Sánchez, Value Considerations in Software Architecture: A Systematic Literature Review, Master thesis degree on Máster Universitario en Software y Sistemas, Universidad Politécnica de Madrid, July 2012.

2.1 Introduction

The first contribution of this thesis, Contribution 1 is *Identification of how software internal quality increases the customer value.* As it is said in Section 1.5.2, to achieve this goal a systematic literature review was performed. Therefore, this chapter follows the structure recommended by Kitchenham and Charters [63] to report systematic literature reviews in software engineering.

The remainder of the chapter is organized as follows: Section 2.2 presents the main concepts involved in the systematic literature review; Section 2.3 details the research method (i.e., a systematic literature review); Section 2.4 discusses the results; and Section 2.5 summarizes the outcomes of the systematic literature review in the context of this thesis.

2.2 Background

2.2.1 Architecting activities

Software architecture is defined as the fundamental concepts or properties of a system in its environment, which are embodied in its elements, relationships, and in the principles of its design and evolution [2]. Software systems are constructed to satisfy customers' and organizations' business goals. According to Bass, Clements, and Kazman [11, ch. 1], the path from abstract goals (i.e. business goals) to concrete systems can be complex. Nevertheless, architecture can help tame this complexity because it is a bridge between the business goals and the final system.

To achieve the goal described in the introduction, it is necessary to clarify what "architecting activities" are performed in software architecture: this term represents the tasks (or activities) that are performed in the entire software life cycle, which are related to software architecture creation or maintenance. Several authors have used different terms to describe these activities.

- Gorton uses *"determine architectural requirement," "architecture design,"* and *"validation"* [45].
- Emery, Hilliard and Rice use *"understand program," "select views," "analyze each view," "integrate views," "trace views to needs,"* and *"validate"* [32].
- Falessi, Cantone and Kruchten use *"requirement analysis," "decision making,"* and *"architectural evaluation"* [33].

- Hofmeister et al. use *"architectural analysis," "architectural synthesis,"* and *"architectural evaluation"* [48].

Using these previous works as references, in this chapter, architecting activities are classified as actions that perform work with a software architecture, in which it is possible to generate value for customers. The actions are defined below:

- ***Analysis*** aims to understand the environment where the system will work. This includes functional and non-functional requirements, schedule restrictions and business requirements.
- ***Synthesis*** or design concerns creating or changing the architecture.
- ***Evaluation*** means testing the architecture to ensure that it satisfies the needs identified in the analysis.
- ***Documentation*** of the architecture includes design decisions, the rationale for such decisions, and management of the knowledge that is relevant to the architecture.
- ***Management*** includes scheduling, tools, methodology, and so on.

2.2.2 Value-Based Software Engineering

In the engineering field "a product or service is generally considered to have good value if that product or service has appropriate performance and cost" [75]. However, because value is subjective [29, p. 22], the concept of customer value is widely used, which is "a customer's perceived preference for, and evaluation of, those product attributes, attribute performances, and consequences arising from use that facilitate (or block) achieving the customer's goals and purposes in use situations" [120]. Therefore, it is possible to claim that value depends on the customers' perceptions and expectations. Moreover, a product will have value for a customer if it satisfies the value propositions of that customer, which represent the customer's expectations. If a product does not fulfill a minimum number of these value propositions, the product will not have value for customers or a subset of them. Therefore, the value for the customers is closely linked to the external quality of the software, that is, the quality that customers can perceive [98].

In software engineering, Value-Based Software Engineering [13] proposes the use of value management in software engineering activities. The goal is to use value as a driver for developing software. Hence, Boehm [14] recommended that software engineering has the goal of adding value, including value-based software architecture.

The focus of this chapter is on value-based software architecture, and therefore it does not investigate the other areas in this field. Value-based architecture involves the reconciliation of the system's objectives with achievable architectural solutions [14].

It can be concluded that value is a complex concept that can be seen from different perspectives. These perspectives have to be taken into account conducting a deep analysis of value.

2.3 Research Method

The method selected to achieve the goals of this chapter is the *Systematic Literature Review* (SLR) [63]. This method was selected because it has been reported to be useful in identifying and summarizing the existing evidence concerning a treatment or technology. A systematic literature review helps to identify any gaps in the current research and suggest areas for further investigation. It also can be used to provide a framework or background for appropriately positioning new research activities [63]. These capacities match very well the goal of this chapter.

The guide published by Kitchenham and Charters [63] was used. The first step in a systematic literature review is to define the review protocol. The following subsections then summarize the protocol used to conduct the literature review.

2.3.1 Research questions

It is necessary to highlight that the objective is focused on internal quality. It was decided to analyze value and architecture without limiting the search to internal quality because the most of the publications do not focus on internal or external quality of software. Therefore, the focus on the internal quality was used in the analysis of the studies, but not in searching them.

The research questions were:

- **RQ1**: What concepts are involved in the value creation in architecting activities (see Section 2.2.1)?
  - **RQ1.1**: In which architecting activities are value considerations taken into account?
  - **RQ1.2**: What motivations and/or goals have driven the use of value-based approaches in software architecting activities?

  - **RQ1.3**: What architecting techniques are value driven, and how do they make use of value?

- **RQ2**: How do architecting activities (see Section 2.2.1) create value?

Following [63], the population, intervention, and outcomes that were used to construct the search string are described below. These three elements helped formulate the research question within the context of the study. *Population* refers to the group of persons under analysis; *intervention* refers to the activities that are analyzed; *outcomes* refers to what results are expected from the studies.

- **Population**: software architecture practitioners and researchers.
- **Intervention**: software architecting activities such as synthesis, evaluation, documentation, and so on.
- **Outcomes of relevance**: how value is used in the context of software architecture.

2.3.2 Search strategy

The search string used to select the relevant studies was created from the research questions. The structure of the search string obtained from this first step is the following.

Listing 2.1: Query string structure

```
(software) and (architecture terms) and (value terms)
```

To identify the terms used in the final string, a former systematic mapping of value-based software engineering [57] was used as a starting point. To help extract of the most relevant terms and inspired by the suggestions about objective search string elicitation provided by Zhang and Ali Babar [127], a data mining tool (RapidMiner [4]) was used to analyze the metadata of the references previously identified by that systematic mapping. The resulting search string can be seen in Listing 2.2. It was necessary to slightly adapt this string so that it could be used in the search engines of the following electronic libraries: IEEExplore, ACM Digital library, Scopus, ScienceDirect, Engineering Village and Web of Knowledge.

Listing 2.2: Query string

```
(("software")
 AND
 ("architecture" OR "highlevel design" OR
"abstract design" OR "solution domain" OR
"description of the system" OR
"system description" OR
"system's description" OR "software system" OR
" architectural")
AND
("value" OR "valuation" OR
 " intellectual property valuation" OR
 "cost benefit" OR "benefit realization" OR
 "business case analysis" OR "economic value" OR
 "economic profit" OR "economic−driven" OR
 "economic driven" OR "return on investment" OR
 "return investment" OR "stakeholder win−win" OR
 "decision multiple criteria" OR
 "competitive position" OR
 "network externalities" OR
 " differentiation value" OR
 "value driven" OR "value−driven" OR
 "drive by value" OR "driven by value" OR
 "business value" OR "business−value" OR
 "value for the business" OR
 "customer value" OR "customer−value" OR
 "added value" OR "added−value" OR
 "perceived value" OR "perceived−value"))
```

To perform the systematic literature review two complementary search processes were performed. The first one included most of the studies, and the second one was performed to update the analysis once the main block of studies had been analyzed previously. Consequently, this review is based on publications to 13 May 2014, which is the date of the last search. The results of the searches were stored in files using the *BibTeX* format. Finally, all references were stored in the Mendeley reference manager [3], which was extensively used in the selection process described below.

2.3.3 Study selection criteria

To select the candidate studies, a set of inclusion and exclusion criteria and a process of applying them were established.

2.3.3.1 Inclusion criteria

- Study refers to the use of value in software architecting activities.
- Study must be peer reviewed (i.e., published in a journal, conference or workshop).

- Study presents evidence or evaluation of the use of value in software architecture.

2.3.3.2 Exclusion criteria

- Study makes reference to the use of value but not in the software architecture context.
- Study is not accessible in full-text format (electronic or physical).
- Study is not a primary study.
- Study does not have the required quality (see Section 2.3.5).

2.3.3.3 Selection process

To apply these criteria the process shown in Figure 2.1 was followed. The filters in process correspond to the step of the process in which studies were discarded. *Filter1* discards papers duplicated or already analyzed, *Filter2* discards papers not peer reviewed for publication in journals, workshops, conferences, and so on, or not primary studies. *Filter3* discards papers clearly not related to the subject of the review, and *Filter4* discards papers not related to the research questions. Finally, *Filter5* discards studies that do not have a sufficiently high-quality score, as described in Section 2.3.5.

2.3.4 Included and excluded studies

The first step consisted of using the search string in all the selected digital libraries. For later consultation, all results were stored in files in the *BibTeX* format. This format was selected because it is often used by many reference managers. The *BibTeX* format facilitates the usage of the obtained references. All references were then stored in the reference manager: Mendeley [3]. The next step was to identify publications selected several times, that is, repeated. A tool provided by Mendeley was very helpful in this step. After removing duplicates, the complete selection process described in Figure 2.1 was performed in the remaining studies. All steps were documented using the tag system provided by the Mendeley reference manager. Tables 2.1 and 2.2 show the numbers of articles analyzed in each step. Remarkably, in many cases, the same article was found in more than one library. As Table 2.2 shows, 58 publications were selected for the systematic literature review.

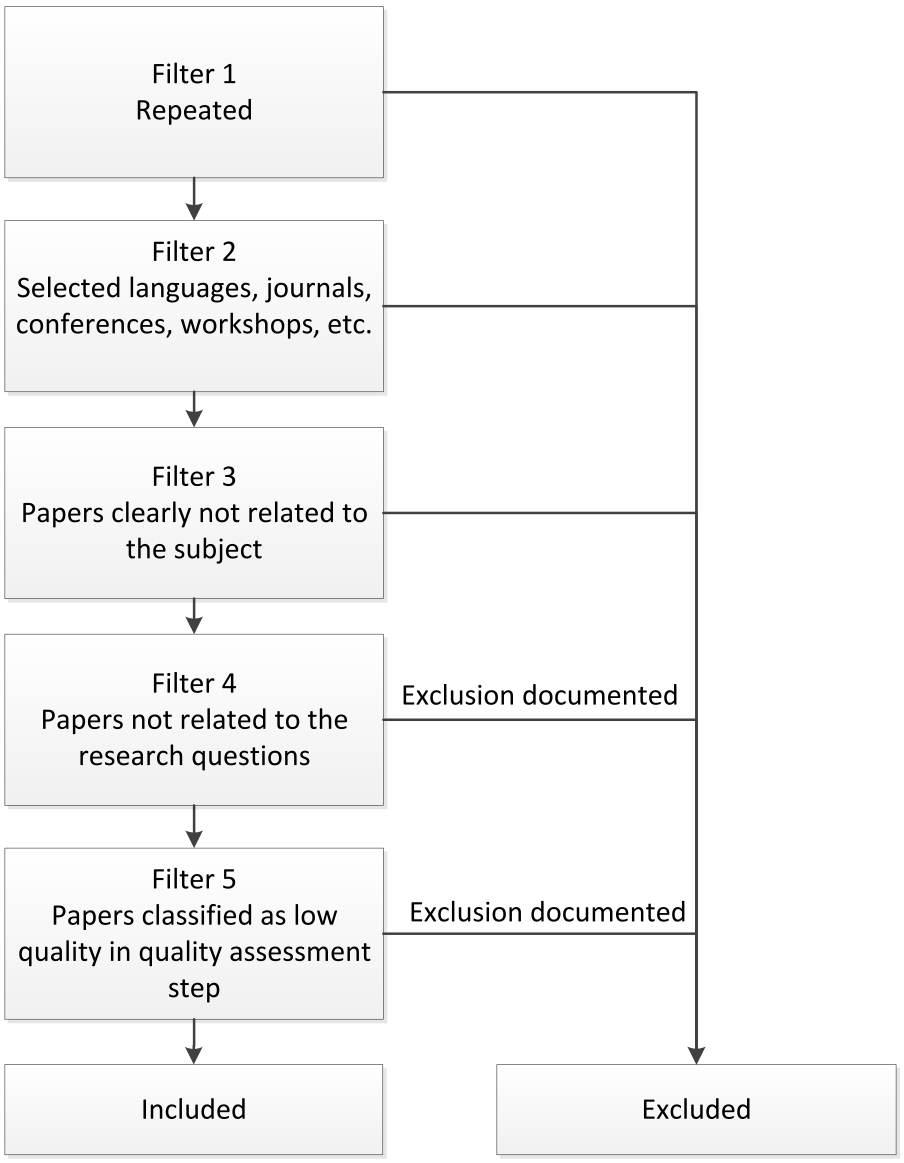

Filter 1
Repeated
Filter 2
Selected languages, journals, conferences, workshops, etc.
Filter 3
Papers clearly not related to the subject
Filter 4
Papers not related to the research questions
Exclusion documented
Filter 5
Papers classified as low quality in quality assessment step
Exclusion documented
Included
Excluded

Figure 2.1: Study selection process

Table 2.1: Papers selected from each database

Database	Found	After Filter 1& 2	After Filter 3	After Filter 4	After Filter 5
IEEExplore	1461	1428	166	28	21
ACM Digital library	744	730	111	23	18
Scopus	4218	4072	262	54	41
ScienceDirect	348	331	22	6	6
Engineering Village	2283	2213	206	47	37
ISI Web of Knowledge	1123	1211	143	30	25

Table 2.2: Total number of found references

Found	After Filter 1 & 2	After Filter 3	After Filter 4	After Filter 5
10177	6683	410	75	58

2.3.5 Assessment of study quality

The quality of the selected studies is an essential factor, first deciding whether a candidate study will be selected, second, it can be used as a tool to assist in the data analysis and synthesis stages. The data about the quality of the selected studies were obtained at the same time as the main data extraction activity was performed.

To conduct the most objective analysis of the studies, a set of questions about quality were chosen and described. These questions were answered using the information extracted from the studies. Hence, all the studies were evaluated following the same criteria.

A limitation of this method is that studies could only be evaluated using the information obtained from the publications. If other additional information, such as technical reports, web pages and so on was available, it was also used. However, because of limited space, many publications could not provide all possible information about the study. Therefore, in practice, this method was an assessment of the available information, rather than an assessment of the quality of the study performed by the authors.

The answers to these questions were stored according to each selected study, using Microsoft Excel documents. These results consist of a score that follows the following rules. For each secondary question, a score between one and four is assigned. Each primary question score is composed based on the mean of all the secondary questions that it includes. Finally, the total score is the sum of all the primary questions. Therefore, the total score is between 11 and 44 because there are eleven primary questions. Finally, the higher the score is, the higher the quality of the information provided by the selected study, which is determined by the authors of this work.

The questions about quality that are addressed by this study can be seen in the Annex B.

The quality score of the selected publications was used to discard papers that obtained a low-quality score so that conclusions could be obtained from the publications having more detailed information. The criteria used considered that studies with scores lower than 22 did not provide enough information. Scores higher than 33 indicated studies that provided detailed information. Publications scoring between 22 and 33 were considered not to have enough information. Figure 2.2 shows the distribution of the quality score of the publications.

To avoid conclusions based on the data extracted from studies that do not provide enough detailed information, according to the described grading process, studies with low-quality scores were discarded in *Filter5* of the selection process described in Section 2.3.3.3. Fifty-eight primary studies fulfilled the quality criteria and therefore were used to answer the research questions.

2.3.6 Data extraction strategy

A form for data extraction was created to extract the information that was relevant to the research questions of this chapter. The goal of this form was to help extract all the required information from each of the selected studies.

The research questions (Section 2.3.1), are focused on identifying concepts and techniques related to software architecture that consider value in some way. The data form used can be seen in Annex A.

Because the selected studies are heterogeneous, they did not follow a common pattern in describing the relationships between value and architecture. Therefore, it was not always possible to obtain input for all topics on the data form from every study. When the required data were obtained from the studies, they were used as the start point in the process of synthesis in order to identify relationships between concepts. The synthesis process is explained in the next section.

2.3.7 Data synthesis process

A narrative synthesis approach was followed to uncover the concepts used in the analyzed publications and the relationships between those concepts. A definition of narrative synthesis is given in [86]: “Narrative synthesis refers to an approach to the systematic review and synthesis of findings from multiple studies that relies primarily on the use of words and text to summarize and explain the findings of the synthesis. Whilst narrative synthesis can involve the manipulation of statistical data, the defining characteristic is that it adopts a textual approach to the process of synthesis to 'tell the story' of the findings from the included studies. As used here 'narrative synthesis' refers to a process of synthesis that can be used in systematic reviews focusing on a wide range of questions, not only those relating to the effectiveness of a particular intervention". This method was selected because it is suitable for heterogeneous studies, such as studies included in the present systematic literature review. The use of narrative synthesis in software engineering was previously reported [27]. Popay et al. [86] produced a guide for using narrative synthesis in systematic literature reviews. Although the guide is oriented towards medical studies, it is suitable for software engineering [27]. In the guide, four steps are proposed to perform a narrative synthesis. These are described below, with suggestions for the possible uses of each:

1. *Developing a theory.* This section corresponds to an initial description of the fundamental concepts, processes, and activities that represent the basis of the studies analyzed.

2. *Developing a preliminary synthesis.* This is the initial analysis of the studies. The goal of this stage is to provide a preliminary view of the studies analyzed and to show factors that could influence the included studies.

3. *Exploring relationships within and between studies.* Studies are analyzed with the goal of revealing relationships between the concepts that they use. The goal of this stage is to show how the concepts are connected and to determine factors that differentiate the studies. This stage is very important because it reveals the concepts that underlie all the selected studies.

4. *Assessing the robustness of the synthesis.* In this part of the synthesis, all the means that have been used to minimize bias should be described.

Figure 2.3 shows the process followed and the techniques used. The first step corresponds to the analysis of the main concepts involved in this study, that is, the analysis of architecting activities and value-based software engineering. The output of this step is summarized in the background section of this paper. For steps two and three, several techniques were used to consider which were the most suitable to answer the research questions. *Tabulations* and *charts* were used to show information in a schematically, making it easier to understand and visualize. Microsoft Excel was used

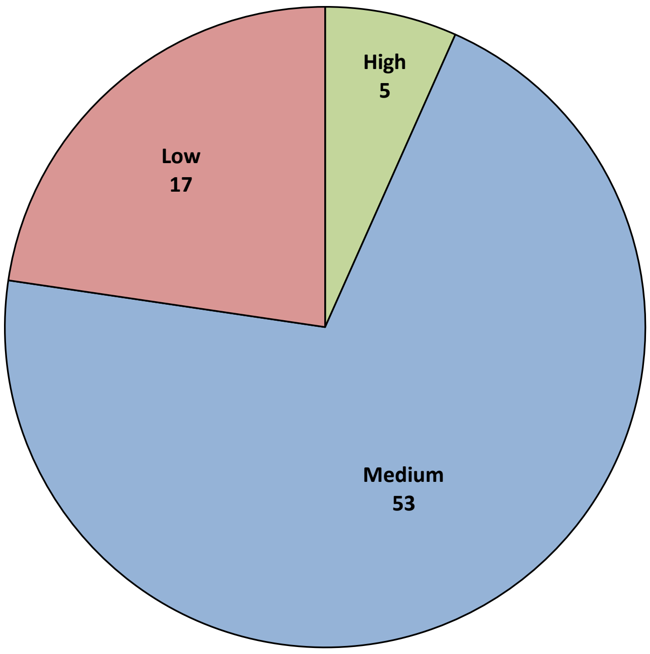


Figure 2.2: Studies quality distribution

to apply these techniques. *Ideas webbing/conceptual mapping* was used to identify the principal concepts in the studies and to determine the relationships between them, including the relationships among the concepts identified in different studies. In this case, Microsoft Visio was used.

Grouping and clustering techniques were used to identify groups of studies and concepts with common characteristics, allowing the classification of the studies to be analyzed. This was done by adding labels to the data collected in the Microsoft Excel sheets that were used to perform the analysis. *Textual descriptions* were used to discuss the underlying implications of the discovered concepts and relationships. The output of steps two and three of the data synthesis are summarized in Section 2.4. Finally, because of the kind of studies analyzed and their difference from one another (e.g., their scopes, contexts, and goals), it was not possible to use techniques proposed to assess the robustness of narrative synthesis for quantitative studies [86]. Therefore, we used the quality criteria described in Section 2.3.5 of the systematic literature review to avoid studies lacking sufficient information to obtain conclusions.

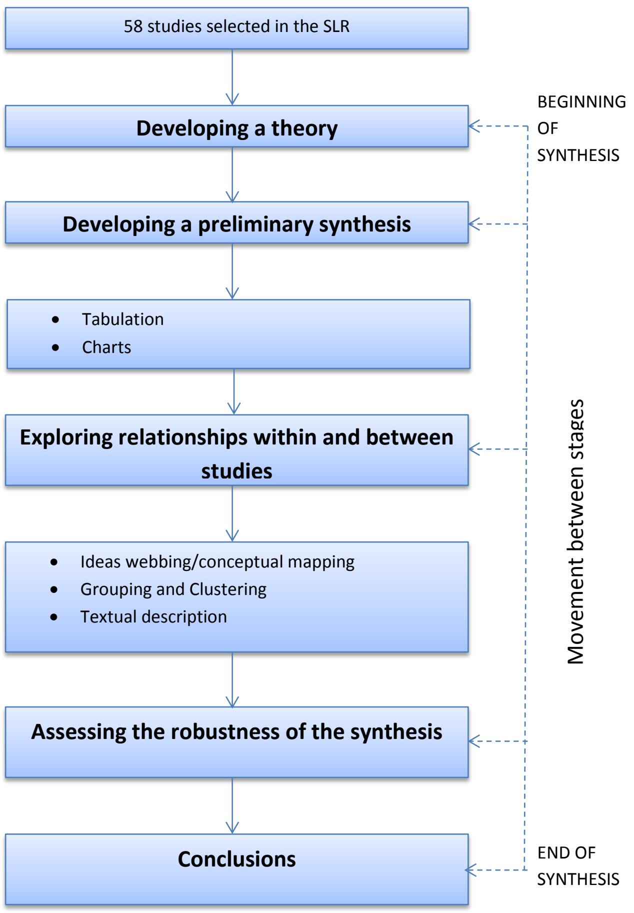

58 studies selected in the SLR
Developing a theory
BEGINNING OF SYNTHESIS
Developing a preliminary synthesis
• Tabulation
• Charts
Exploring relationships within and between studies
• Ideas webbing/conceptual mapping
• Grouping and Clustering
• Textual description
Movement between stages
Assessing the robustness of the synthesis
Conclusions
END OF SYNTHESIS

Figure 2.3: Synthesis process, inspired from [86]

Table 2.3: Architecting activities and the studies where they were found

Architecture activities	References
Synthesis	[S12][S11][S58][S53][S19][S2][S21][S8][S42][S51][S4][S49][S9][S16][S23][S22][S55][S56][S3][S57][S54][S52][S41]
Documentation	[S18][S17][S14][S15]
Evaluation	[S20][S40][S27][S38][S34][S50][S5][S29][S35][S43][S28][S36][S30][S7][S48][S24][S1][S37][S32][S13][S39][S31][S6]
Design as a whole	[S25][S26][S47][S33]
Management	[S44][S10][S46][S45]

2.4 Results

In this section, the results obtained for each research question defined in Section 2.3.1 are described.

2.4.1 RQ1: What concepts are involved in the value creation in architecting activities?

To answer this question focus was on architecting activities in which the value concept was used, as well as on techniques used or proposed to support activities using the value concept. The goal was to obtain a general picture of the current use of value in software architecture. To answer this question, more specific research questions were formulated.

2.4.1.1 RQ1.1 In which architecting activities are value considerations taken into account?

The goal of this question is to identify the architecting activities (see Section 2.2.1) that consider the concept of value and use it as a driver to perform the activity.

Figure 2.4 shows the distribution of the architecting activities identified in the selected studies. Table 2.3 shows studies classified by architecting activity.

Studies not focused on concrete activities but on architecting in general were classified as *Architecting as a whole.*

The results showed that the most frequent activities in which value was used are evaluation and synthesis. Nevertheless, it is necessary to emphasize that all studies involved somehow architectural analysis to a greater or lesser degree; however, this architectural activity is not the sole activity reported in these studies. As architectural analysis aims to understand the environment where the system will work, this means that all the architecting activities take into account the context of software when in such activities value is considered.

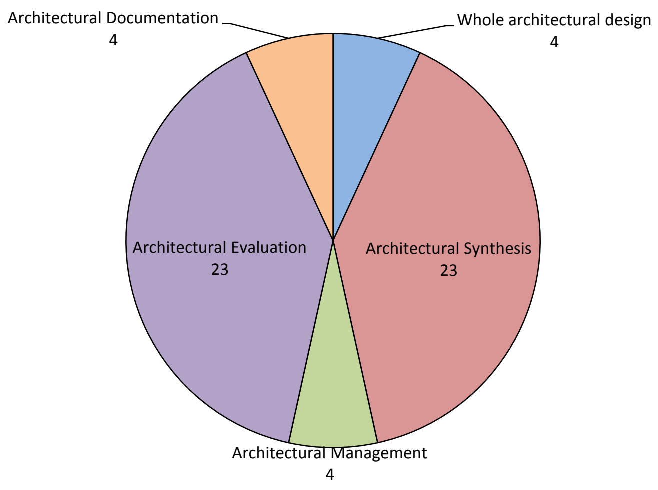


Figure 2.4: Studies included in the study classified by architecting activity

2.4.1.2 RQ1.2 What motivations and/or goals have driven the use value-based approaches in software architecting activities?

Several goals were identified as the motivation to use value-based approaches. Figure 2.5 shows the number of studies classified by identified goals. Table 2.4 shows the selected studies classified by goal. The goals are not bound to a given architecting activity. As shown, *Achieve stakeholders' concerns* is the most frequently cited goal in all the studies. This goal is common to studies about architectural evaluation or decision making in architectural synthesis when the effect of architecture characteristics on stakeholders' concerns is considered. These studies are focused mainly on external quality because they are focused on the concerns that can be perceived by stakeholders. Only if maintainability or flexibility are considered by stakeholders, internal quality would be considered, but without clearly analyzing the impact of the internal quality in the future possible changes.

Another goal that is frequently referenced is *to assess the value of flexibility*. These studies intend to calculate or estimate the value derived from decisions made about software architecture through the *flexibility* that they add to software. These studies deal with internal quality by analyzing how the flexibility of a system will help to accommodate future changes, and therefore, it will help to add value for the customers in the evolution of software.

The third most referenced goal is *to assess the value of architectural decisions*. In many cases, the motivation corresponds to the need to choose between different design

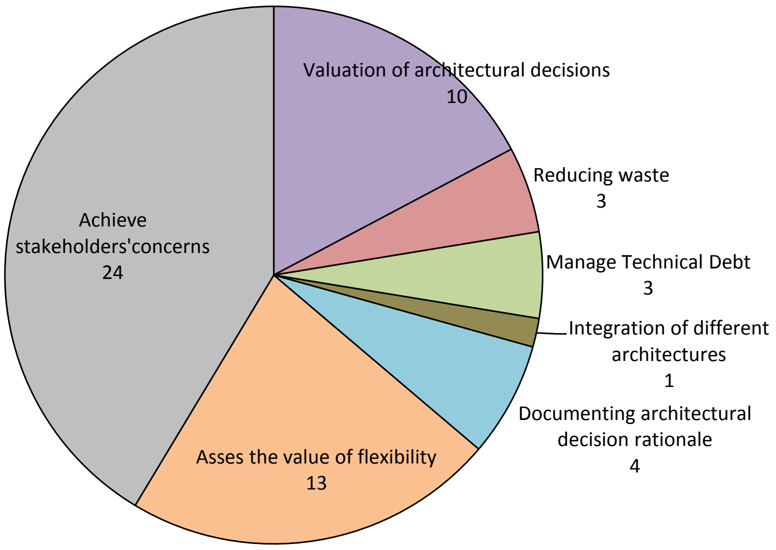


Figure 2.5: Studies' goals related to architecture

Table 2.4: Goals of the selected studies

Goals	References
Achieve stakeholders' concerns	[S20][S27][S34][S47][S2][S21][S29][S51][S35][S43][S28][S4] [S36][S30][S48][S9][S24][S1][S37][S55][S56][S13][S31][S54]
Valuation of architectural decisions	[S12][S40][S25][S26][S53][S8][S49][S39][S57][S52]
Assess the value of flexibility	[S11][S19][S5][S42][S7][S16][S23][S22][S32][S33][S3][S41][S6]
Documenting architectural decision rationale	[S18][S17][S14][S15]
Reducing waste	[S10][S46][S45]
Integration of different architectures	[S58]
Manage Technical Debt	[S38][S44][S50]

options. Hence, to calculate or estimate the value of those options will help in making decisions. The options will have more or less value considering the long-term impact and depending on possible future change scenarios.

To manage technical debt is very close to flexibility and options. In that case, technical debt management is focused on analyzing the relationship between internal quality of the software with future costs and capacity to support future changes. Therefore, technical debt management helps to manage the internal quality and its degradation over the project evolution.

Reducing waste and *integration of different architectures* are special cases of *achieving stakeholders' concerns.*

2.4.1.3 RQ1.3 What architecting techniques are value driven, and how do they make use of value?

The first step to answer this question was to identify the techniques used in the selected studies. Some studies used several techniques, while others did not use any. Table 2.5 shows the identified techniques, as well as the references to the studies where they were identified. It is divided into two main sections, the top one shows the evaluation methods identified. The bottom section of the table shows the remaining techniques used in the analyzed studies.

The goal of architecture evaluation is to analyze software architecture to identify risks and verify that the quality requirements have been addressed in the design of the architecture [30]. The row *I-Evaluation* in Table 2.5 shows the identified evaluation methods and techniques. Data have been aggregated. That is, different versions of the same method or technique were assumed the same. For example, several versions of the Cost Benefit Analysis Method (CBAM) were used in the studies (e.g., CBAM, CBAM2, WinCBAM, CBAM+AHP). In this case, they all have been counted as CBAM. Two frameworks to analyze architectural evaluation methods were used to help in the analysis and synthesis of the methods.

The first framework was defined by Babar et al. [10] and is an extension of the framework defined by Dobrica and Niemela in [30]. The summary of the analysis can be seen in Table 2.6. It is possible to highlight that:

- Some of the methods (ATAM, Losavio et al., SQUASH, SystEM-PLA, and QuaDAI) are focused on the assessment of quality attribute. That is, the objective of these methods is to determine whether architectures will satisfy, to a certain level, different quality attributes (QA). The value is considered by being sure that the architecture satisfies the stakeholders' concerns.

- Other methods (CBAM, LiVASAE, CBAM+AHP, CBAM+ANP, WinCBAM, and REARM) consider return on investment (ROI) in the architecture evaluation. That is, these methods evaluate the benefits and costs of the different architecture strategies. These methods are mainly centered on the economic perspective.

- Finally, MORPHOSIS is centered only on maintainability, that is, on future changes.

The second analysis was performed following the framework defined by Kazman et al. [61]. The summary of the second analysis is shown in Table 2.7. It is possible to highlight that:

Table 2.5: Methods/Techniques used and/or defined. One study can use several methods and techniques. Different versions of the same method or technique have been count as the same

	Used / Defined Methods	No.	References
I-Evaluation	ATAM	12	[S20][S27][S34][S29][S35][S43] [S28][S48][S32][S30][S37][S36]
	CBAM	6	[S27][S34][S29][S35][S43][S30]
	Losavio et al.	2	[S36][S37]
	QuaDAI	1	[S20]
	REARM	1	[S40]
	LiVASAE	1	[S30]
	SystEM-PLA	1	[S48]
	SQUASH	1	[S24]
	SAAM	1	[S1]
	MORPHOSIS	1	[S32]
	HoPLSAA	1	[S48]
	ALMA	1	[S32]
II-Others	Real Options Valuation	12	[S53][S19][S5][S42][S49][S7] [S16][S22][S3][S52][S41][S6]
	DSM	6	[S12][S11][S44][S53][S10][S46]
	Technical Debt	5	[S38][S44][S19][S50][S3]
	AHP	4	[S34][S2][S30][S56]
	GQM	3	[S20][S38][S48]
	ArchOptions	3	[S5][S7][S6]
	Customer value-in-use	2	[S25][S26]
	Lean	2	[S23][S45]
	Customer segments	2	[S25][S26]
	DMM	2	[S10][S46]
	NOV	1	[S11]
	Ojala 2008	1	[S47]
	ArchDesigner	1	[S2]
	Game of involuntary altruism	1	[S8]
	PSO	1	[S51]
	Andrews et al. 2005	1	[S4]
	Diaz-Pace et al. 2013	1	[S14]
	NPV	1	[S57]
	e3-VALUE	1	[S21]
	VSM	1	[S23]
	Svahnberg et al. 2003	1	[S56]
	ACN	1	[S11]
	WinWin	1	[S29]
	CloudMTD	1	[S3]
	ANP	1	[S34]

Table 2.6: Analysis of the evaluation methods used in the selected studies based on the framework described in [10]

	ATAM [60]	CBAM [S27]	Losavio et al. [S37]	SQUASH [51]	LiVASAE [S30]	CBAM+AHP CBAM+ANP [S34]	WinCBAM [S29]	MORPHOSIS [S32]	SystEM-PLA [S48]	REARM [S40]	QuaDAI [S20]
Maturity stage	Development	Development	Dormant	Dormant	Dormant	Dormant	Dormant	Dormant	Dormant	Dormant	Dormant
Process support	Process described	Process described	briefly described	Process described	Briefly described	Process described	Process described	Process described	Process described	Process described	Process described
Method's activities	9 activities in 2 phases	Previous ATAM analysis, 6 steps for triage and 6 steps for detailed analysis	8 steps	2 phases with a total of 7 steps	Previous ATAM analysis & 3 steps	Previous ATAM analysis & 2 phases with a total of 7 steps	Previous ATAM analysis & CBAM analysis & 3 steps	3 phases, the first one has the same steps that ALMA analysis	3 phases, with a total of 14 steps	3 phases	3 phases
Method's goals	Sensitivity & Trade-off analysis	ROI evaluation at architectural level	Quality attribute achievement	Selecting the most relevant scenarios.	ROI evaluation at architectural level	ROI evaluation at architectural level	ROI evaluation at architectural level	Evolution scenarios analysis	Quality attributes and structure analysis of product line architectures	ROI evaluation of reference architecture	Quality attributes evaluation in software product lines
Quality attributes addressed	Multiple attributes	Multiple attributes	ISO-9126-1 quality attributes	Multiple attributes	Multiple attributes	Multiple attributes	Multiple attributes	Maintainability	Multiple attributes	Reuse and error cost	Multiple attributes
Applicable project stage	After SA/detailed design, iterative process	After ATAM analysis.	Early architecture design	Before architecture design	After ATAM analysis	After ATAM analysis	After ATAM analysis	Software maintenance	After product line architecture design	After reference architecture implementation or design	After product architecture derivation or transformation
Architectural description	Architectural views related to the selected scenarios	Left to previous ATAM analysis	Left to users	No needed	Left to previous ATAM analysis	Left to previous ATAM analysis	Left to previous ATAM analysis	It uses source code analysis	It uses UML architecture specification	It is not specified	Models describing architectural views and viewpoints.
Evaluation approaches	Scenario-based analysis & stakeholders consensus	Scenario-based analysis & stakeholders consensus & cost estimation	Metrics and measures of quality attributes	Defining quantitative metrics for QAs.	Scenario-based analysis & stakeholders consensus (AHP) & cost estimation	Scenario-based & pair-wise comparison	Scenario based & stakeholders consensus	Scenario based & maintainability metrics	Scenarios & metrics	Metrics	Metrics
Stakeholders involved	All major stakeholders	All major stakeholders	Not specified	All major stakeholders	All major stakeholders	All major stakeholders	All major stakeholders	Architects & developers	Stakeholders & evaluation team	project manager & architects & developers	Application engineer & Evaluator & Architect
Support for non-technical issues	It is mainly oriented to technical issues (QAs)	It analyzes architecture with economic perspective	It is mainly oriented to technical issues (QAs)	It analyzes QAs, Risk, and Cost.	It analyzes architecture with economic perspective	It analyzes architecture with economic perspective	It analyzes architecture with economic perspective	It analyzes architecture with maintenance cost perspective	It is mainly oriented to technical issues (QAs), it considers cost and risk	It is oriented to cost estimations	It is mainly oriented to technical issues (QAs)
Method's validation	Used in several domains	Used in several domains	Only one example.	Only one context	Only one context	Only one context	One context	One context	One context	One context	One context
Resources required	Evaluation team & stakeholders	Evaluation team & stakeholders	Evaluation team.	Stakeholders & who makes the evaluation.	Stakeholders & Evaluation Team	Stakeholders & Evaluation team	Stakeholders & Evaluation team	Architectural description & source code & Evaluation team	Architectural description & features description	Depending on the selected metrics, it needs different inputs	Product features & Quality attributes priorities & SPL design

- The evaluation methods are mostly based on the evaluation of quality attributes. The value is considered by being sure that the architecture satisfies the stakeholders' concerns.
- Only some of them consider the economic implications (i.e., return on investment) of architecture strategies (see Table 2.7).
- Only MORPHOSIS and REARM explicitly take into account architecture decisions impact in the long term. MORPHOSIS considers future changes with regard to the maintainability of the architecture, and REARM considers reuse in the context of reference architectures.
- Although identifying business goals is key in determining whether an architecture provides value, several methods do not provide the means to do it.

Out of evaluation of architectures (see row *II-Others* of Table 2.5), Real Options Valuation [49] was frequently used in 12 of the selected studies. Real Options is based on the long-term impact of decisions that are made at architectural level in software projects. Studies that use Real Options are centered the most often on either i) assessing the value of different design alternatives (choosing the most suitable for the analyzed situation) or ii) assessing the value of the flexibility added by the architecture or its design. In both cases, the motivation for using real options is to have adequate tools, based on economic aspects, to assess the value of design decisions taking into account future changes. But it is necessary to decide how much effort is profitable to use for adding value in the short term (increasing the external quality of products) and in the long term (providing flexibility to be ready for future changes). The studies that use the concept of technical debt (see Table 2.5) are mainly focused on this decision. However, currently, there is no well-established approach to using real options in making decisions at software architectural level.

2.4.1.4 RQ1 Conclusions

As a final conclusion of the RQ1, it is possible to synthesize the RQ1.1, RQ1.2, and RQ1.3 in a set of concepts that are involved in the value creation from the architecting activities. These concepts are summarized in Table 2.8.

2.4.2 RQ2: How do architecting activities create value?

Methods and techniques used to evaluate software architecture (see Tables 2.6 and 2.7) are mainly centered on the relationship between architectural strategies and stake-

Table 2.7: Analysis of the evaluation methods based on Kazman et al. criteria [61]

	ATAM [60]	CBAM [S27]	Losavio et al. [S37]	SQUASH [51]	LiVASAE [S30]	CBAM+AHP CBAM+ANP [S34]	WinCBAM [S29]	MORPHOSIS [S32]	SystEM-PLA [S48]	REARM [S40]	QuaDAI [S20]
Context and goal identification	It has specific phases to identify them	Left to previous ATAM analysis	It is focused on QAs (ISO-9126-1) but it does not link them with business goals	questioning to the stakeholders but it is very related to QA.	Left to previous ATAM analysis	Left to previous ATAM analysis	It uses the previous ATAM analysis and complete it with WinWin requirement negotiation method	It uses evolution scenarios	It defines metrics using GQM	To determine the ROI of a reference architecture	It analyze QAs in SPL to determine variation points and architecture transformation to achieve the QAs expected level
Focus and properties under examination	Quality attributes and scenarios to analyze them	It makes relationships between ASs and QAs to keep the focus on the attributes analyzed	Prioritizing QAs (ISO-9126-1) helps to focus on the most relevant for the project	Several quality attributes	Several attributes	It makes relationships between ASs and QAs to keep the focus on the attributes analyzed	It makes relationships between ASs and QAs to keep the focus on the attributes analyzed	It uses maintainability according to ISO/IEC 25010	Quality attributes and scenarios to analyze them	Reusability and Errors	It is focused on analyzing multiple QAs
Analysis support	It uses scenarios to lead the analysis	It uses scenarios and architectural strategies to lead the analysis	It provides metrics to analyze the different QAs (ISO-9126-1) with an architecture perspective	It uses scenarios and metrics to analyze QA. And always taking into account risk and cost.	It uses scenarios and AHP for evaluating architecture against QA selected.	It uses scenarios and AHP o ANP	It is based on the combination of ATAM, CBAM, and WinWin	It uses code metrics to measure maintainability	Using GQM defines metrics for the QAs	It uses metrics for measure historical data and estimations for future situations	It is based on architectural views and viewpoints.
Determining analysis outcomes	Risks identified are attached to business goals.	Adding economic perspective to ATAM outcomes.	little support for tracing the effects of business goals through an architecture analysis	Little support for tracing the effects of business goals through architecture analysis	Adding economic perspective to ATAM outcomes.	Adding economic perspective to ATAM outcomes.	Adding economic perspective to ATAM outcomes.	It identifies architecture refinements to improve the capacity to evolve of an architecture	It uses GQM to trace QAs support with concrete metrics	It provides economical information about the usage of a reference architecture	It provides information about when it is necessary to create variation points or perform architectural transformation to achieve the quality expected.

Table 2.8: Concepts involved in the value creation from the architecting activities

Concept	RQs	Description
Context	RQ1.1	To consider value, it is necessary to take into account the context of software when architecting activities are performed.
Stakeholders' Concerns	RQ1.2 RQ1.3	The most direct way to add value from architecture is by creating architectures that satisfy the stakeholders' concerns. This is deals mainly with external quality of software
Long-term impact	RQ1.2 RQ1.3	The long-term impact of the decisions made in architecture can be used to analyze the value of internal quality of software
Flexibility	RQ1.2 RQ1.3	Flexibility is a concept used to describe the capacity of software to accommodate future changes. Therefore, it is related to the internal quality of the software because it is not perceived by the customers.
Technical debt	RQ1.2 RQ1.3	Technical debt can be used to make trade-offs between the internal and external quality of software
Return on investment	RQ1.3	To make decisions based on value, decisions have to be driven by return on investment
Technical and business goals	RQ1.3	To use value as a driver, technical and business goals have to be conciliated.

holders' concerns. This same focus is used in studies on architecture synthesis, which calculate the benefit of a specific architecture or a change aimed to improve quality attributes. Stakeholders' concerns represent the interest of the stakeholders. Only by satisfying those interests will the system add value to the stakeholders, being the stakeholders all those who have some interest in the system. Architectural strategies represent the decisions that are made when the architecture is designed or evolved.

The goal is to determine which architectural strategies are the most relevant with respect to the stakeholders' concerns. This information is usually stakeholder-based, and therefore subjective. Subjectivity adds uncertainty because different stakeholders have different perceptions of the concerns prioritization. Therefore, it could be necessary to make trade-offs between the concerns of the stakeholders.

However, this does not indicate how an architecture satisfies future value propositions or changes in the current ones. Brown, Nord, Ozkaya, and Pais, studied the dependencies between architectural elements and capabilities of the system [S10, S45]. They affirmed that by focusing on generating the maximum value as soon as possible, higher costs could be generated in the long term because of the need for more refactoring tasks. In that perspective on the long-term, Real Options [S52][S6][S5][S7] are used to value architecture flexibility. They argued that a system with enough flexibility would support changes more easily. The trade-off between short-term value and long-term costs gives value to flexibility. Therefore decisions have to be made to decide when it would be profitable to invest in improving flexibility by modifying the system architecture. That is, to take into account the long term does not mean expending great effort in development to accommodate future changes, but considering how much effort to expend.

Several studies used the concept of architectural technical debt to describe the negative effect that poor architecture has on the future evolution or operation of a system. In [S44], technical debt was used to identify a better combination of features to be implemented in a release and the order implementation of such features. In [S19] technical debt was used to select the most profitable level of flexibility in an architecture. Hence, it was possible to estimate whether it is profitable to refactor an architecture or not. In [S38][S50], technical debt was used to ensure the internal quality of the architecture. That is, the capacity of the architecture to support the future maintenance and evolution of the system. In general, studies about technical debt were highly related with the value of being prepared for future changes. In this sense, technical debt can be used to manage how much effort is necessary to prepare for future changes. Hence, technical debt should be considered in deciding whether it is necessary to refactor a software to avoid excessive maintenance costs in the future.

In [S10][S45][S44] techniques for selecting the most optimal release path to develop a software product are presented. Those techniques are used to select the most worthwhile release plan taking into account value and cost of change incurred by refactoring the architecture or by generating technical debt. Hence, those studies showed that in determining value, it is necessary to take into account the possible future changes and the capacity to accommodate them (flexibility). The studies showed how that value at the architectural level has to be considered in not only design decisions but also issues, such as the order of feature implementations. This order can highly affect the cost of software development caused by the need to refactor. Therefore, it is necessary to synchronize the decisions made at the project management level with the available information about the architecture.

2.4.2.1 RQ2 Conclusions

As a final conclusion of the RQ2, it is possible to synthesize a set of actions that are involved in the value creation from the architecting activities. These actions are summarized in Table 2.9.

Table 2.9: Actions involved in the value creation from the architecting activities

Actions	Description
Satisfy stakeholders' concerns	Prioritization of concern and determination of architectural strategies to satisfy them
Decisions about flexibility investments	Trade-offs between short and long-term value and costs of changes focused on improving the changeability of software

2.5 Conclusions

This chapter aimed to determine how value can be created from software architecture. To accomplish this goal a systematic literature review was performed to analyze extensively the available studies on the subject. Fifty-eight studies were identified as considering value at the level of software architecture.

The most direct way to add value is by implementing architectural strategies that satisfy the stakeholders' concerns. But this way is mainly focused on adding external quality to software. Focusing on internal quality, several authors provided evidence that the order in which architectural strategies for satisfying concerns are implemented is significant. The order can highly influence the cost of system development because the amount and complexity of refactoring may increase for future changes.

Also focusing on internal quality, it is essential to consider both short-term and long-term impact of architectural strategies. Hence, architectures should be analyzed from the perspective of how they would perform in an uncertain future. This does not mean that architectures should be implemented to accommodate all possible future changes. However, it is important to estimate when it would be profitable to develop architectural strategies that anticipate future changes or to implement the simplest solutions. In designing an architecture that is open to changes extra costs will usually be incurred; conversely, the simplest solution usually implies lower costs in the short term, which is also valuable. That is, decisions about the flexibility of architecture have to be made. These decisions are based on a trade-off between short-term value and long-term costs.

Architectural technical debt refers to the cost of internal quality weaknesses at the level of architectural design. Technical debt should be managed to control the cost of the maintenance and operation of a system. Therefore, this variable should be considered in any architecture analysis from a long-term perspective. The extra cost caused by technical debt may reduce the value of a product in the future. Therefore, the concept of technical debt helps to identify the value of flexibility.

One way to increase the flexibility is by option creation and Real Option valuation.

Having options will help deal with future changes. There are several ways to create options at the architectural level: modularity, variability, and any other way that generates alternatives for implement different architectural strategies. Having options has value at the expense of additional cost.

2.6 Selected Publications

See Annex C

Chapter 3

Preliminary Case Study on Technical Debt Management

The goal of this chapter is to put into practice the concepts, identified in Chapter 2, to add value from the internal quality of software. To do that, a preliminary case study was performed with the objective of obtaining further knowledge about how these concepts can be effectively used in software projects. This chapter is an excerpt from the following papers:

- Fernández Sánchez, Carlos; Díaz Fernández, Jessica; Garbajosa Sopeña, Juan y Pérez Benedí, Jennifer (2013). A Cost-Benefit analysis model for technical debt management considering uncertainty and time. Work in progress session at the 39th Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2013) Santander, Spain, September 4-6, 2013. [38].
- C. Fernández-Sánchez, J. Díaz, J. Pérez and J. Garbajosa, Guiding Flexibility Investment in Agile Architecting. In proceeding of the 47th Hawaii International Conference on System Sciences, Waikoloa, HI, 2014, pp. 4807-4816. [35].

3.1 Introduction

This chapter complements Chapter 2 to complete the first contribution of this thesis: Contribution 1, *Identification of how software internal quality increases the customer value*. This chapter presents a model for making decisions about flexibility investments taking into account the outcomes of Chapter 2. Once the model was defined, it was used in a preliminary case study to extract first impressions of its limitations and advantages for using it in real projects. Therefore, the case study is also presented in this chapter.

The remainder of the chapter is organized as follows: Section 3.2 presents the main concepts used in the chapter; Section 3.3 presents a model for cost-benefit analysis of technical debt; Section 3.4 details how a case study to use the model was designed and planned; Section 3.5 describes the execution of the case study; Section 3.6 discusses the results; Section 3.7 identifies limitations of the case study; and Section 3.8 summarizes the outcomes of the case study in the context of this thesis.

3.2 Background

In Chapter 2 several concepts and actions were highlighted for being involved in value creation in software from architecting activities. These concepts and actions were used to define a model to perform a cost-benefit analysis of decisions about improving the internal quality of software.

Focused on software internal quality, the main action that was identified is to make decisions about flexibility investments (see Table 2.9). That is, it is necessary to make tradeoffs between short and long term impact of changes focused on improving the capacity of software to accommodate changes. The main identified concept that is involved in this tradeoff is technical debt (see Table 2.8).

Technical debt has been used as a useful means for making the intrinsic cost of the internal software quality weaknesses visible [64]. Specifically, technical debt is expressed in terms of two main concepts: principal and interest [114]. The principal is the cost of eliminating (or reducing) the impact of a, so called, technical debt item in a software system; whereas the interest is the recurring cost, over a time period, of not eliminating a technical debt item.

Technical debt management consists of identifying the sources of the extra costs of software maintenance and evolution and determining whether it is profitable to invest efforts into improving a software system [114]. Therefore for managing technical debt, it is necessary to estimate the principal and the interest and performing a cost-benefit analysis considering both. This cost-benefit analysis allows one to determine if to remove technical debt is profitable and to prioritize which items incurring technical debt should be fixed first [124].

Thus, by managing technical debt, it is possible to consider the long-term impact of bad internal quality in future changes as well as the cost of improving the internal quality. To consider the long term it is necessary to take into account the time frame, and consequently, the possible evolution of the interest and the uncertainty involved in that evolution. The time frame refers to the time period under study. This time frame could be determined by external constraints or deadlines such as legal normative issues that an application should be bound to, milestones or, availability of resources or contractual restrictions, among others. Due to the interest is a cost that must be paid continuously over time, it will cause accumulated costs. Therefore, with enough time ahead, paying off the principal is always profitable because the accumulated interest grows and grows. Hence, determining the time frame is important for a realistic cost-benefit analysis. Considering this it becomes possible to depict a number of probable future scenarios.

Due to the uncertainty involved in software evolution, the interest, the cost of not removing technical debt, is associated with a probability of being paid. Interest probability is the probability that no extra cost is derived from technical debt. For example, if the interest for a technical debt item is estimated in terms of extra maintenance cost, but the software system incurring technical debt need not be changed over a period of time, then no interest has to be paid during this period.

This chapter presents a model to perform cost-benefit analyses that are based on technical debt and that deals with interest probability and considers the time frame under analysis. This model has been used in a case study. Findings and conclusions about how to use technical debt management in projects are presented. Finally, it is discussed why technical debt has to be considered from a value-based software engineering perspective.

3.3 Modeling Technical Debt Considering the Interest Probability

This section presents a model to perform a cost-benefit analysis based on the technical debt concept, including principal, interest, and interest probability in which the time frame is a model variable. The goal of this model is to be able to use technical debt in decision making about internal quality of software by considering the long term and the ROI of such decisions. That goal is derived from the concepts to add value from the internal quality of software that were identified in Chapter 2 (see Table 2.8). This model takes advantage of decision trees [105, 112]. Decision trees are used to estimate the expected value of the interest considering its probability. Then, to obtain the net expected value of paying off the principal, the principal is subtracted from the interest expected value. That is, the principal is considered the cost of removing technical debt, and the avoided interest is considered the benefit of removing technical debt. Decision trees have been selected because they facilitate the understanding of the technical debt of the system under analysis. This is due to the fact that they can be used to illustrate the possible evolution paths by assigning probabilities to their branches and assigning a weight to each path. That is, tree branches graphically represent several alternatives for the interest evolution. As a result, decision trees usage facilitates the understanding of the technical debt concept of a system under analysis. More complex techniques would make it harder to understand the technical debt behavior.

Figure 3.1 shows an example of a decision tree, in this case, for simplicity reasons, a binary decision tree. Decision trees are used in the model for estimating the interest probability (see Figure 3.1). The tree grows over the time frame under analysis (see Figure 3.1). The time frame is divided into periods of time, in such a way the interest can evolve from the current period of time to the next one. Branching coincides with the time frame events ($t_0 \dots t_n$), over which the interest evolves. The root node labeled as $Interest_0$ represents the current estimated interest at the moment of the analysis. In $Period1$, there are two nodes that represent the interest in that moment, one pessimistic labeled as $Interest_{1,1}$, and another one optimistic labeled as $Interest_{1,2}$. The lines that join the nodes represent the possible paths in the evolution of the interest and are labeled with the probability of such evolution. For example, there is a line between the node $Interest_0$ and $Interest_{1,1}$ that indicates that the interest can evolve following that path, and the probability $p_{1,1}$ is the probability of such evolution (see Figure 3.1). Finally, the number of depth levels of the tree is established by the number of time periods defined.

To represent the accumulated interest during the time frame under study is also required. Decision trees are also useful to represent this. From the tree in Figure 3.1, it is possible to derive a new decision tree with the accumulated interest ($AccInt$) in their nodes. This new tree has the same probabilities, periods and structure as the one shown in Figure 3.1. This tree is named the accumulated interest evolution tree (see Figure 3.2).

The accumulated interest evolution tree provides a key data for the study; its leaf nodes represent the possible results of the interest evolution (see Figure 3.2). This

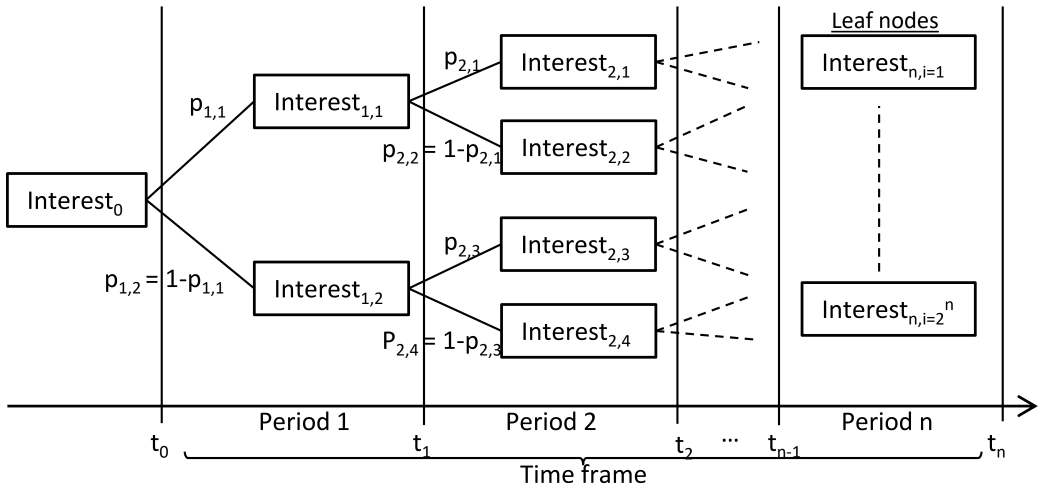


Figure 3.1: Modeling interest probability using decision trees

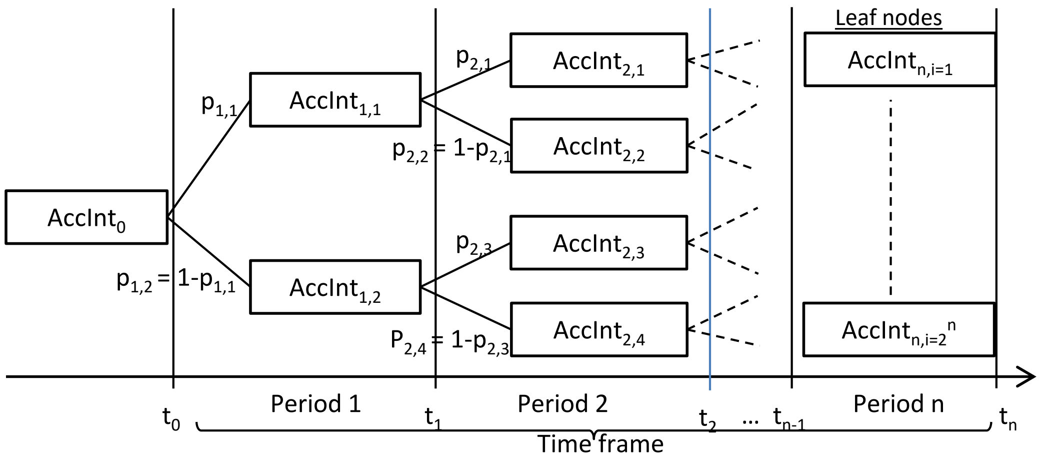


Figure 3.2: Accumulated interest evolution tree

data is necessary to calculate the expected value (EV) of paying off the principal by calculating the expected interest that would be avoided. As a result, net expected value (NEV) can be calculated using this data. To calculate the EV, it is necessary to sum all the leaf nodes of the accumulated interest evolution tree ($AccInt_n$) weighted by their probability. These probabilities are the combined probability (CP) of the whole branch. Let n be the depth of the tree, i the number of the node into a level, and $p_{n,i}$ the probability that the node $AccInt_{n,i}$ occurs from its parent node, the EV is calculated with Equations 3.1 and 3.2.

$$\begin{cases} CP_{n,i} = p_{n,i} & \text{if } n = 1 \\ CP_{n,i} = p_{n,i} * CP_{n-1,\frac{i+i mod 2}{2}} & \text{if } n > 1 \end{cases} \tag{3.1}$$

$$EV = \sum_{i=1}^{2^n} AccInt_{n,i} * CP_{n,i} \tag{3.2}$$

Finally, the net expected value (NEV) of paying off the debt is calculated by subtracting the principal ($Principal$) to the expected value (EV) (see Equation 3.3).

$$NEV = EV - Principal \tag{3.3}$$

As a result, the cost-benefit analysis model obtains an estimation of the NEV considering the interest probability, the time frame and the cost of paying off the principal.

Decision trees seem adequate to model the technical debt of software and to understand how interest accumulates over time. The model has been formalized using binary trees for simplicity, but if needed more complex decision trees can be used. To get experience from the use of the model in large projects, it was used in a case study that is described in the following sections of this chapter.

3.4 Case Study Design and Planning

This section presents the case study in which the previously defined model was used. The guide of Runeson and Höst [94] was followed to conduct this exploratory case study. The next subsections are organized following the recommendation of the mentioned guide.

3.4.1 Objectives

The goal of this case study was to find advantages and limits when managing technical debt using the model defined in Section 3.3. By putting into practice the model it was expected to obtain interesting findings and insights to be able to make conclusions about the applicability of the model in software development projects.

3.4.2 Rationale

The model described in Section 3.3 was intended to be used to perform a cost-benefit analysis considering the principal, interest, interest probability, and time frame. In this case study, this model was analyzed in the context of using it for making decisions about improving the internal quality of software. Specifically, the case study was focused on improving the flexibility of software. Flexibility is the capacity of a software to accommodate new changes, and therefore, it is close to technical debt, because technical debt makes reference to the capacity of software to be maintained and evolved [64].

3.4.3 Case and Subjects Selection

The case study was conducted in an experimental i-smart software factory (iSSF) [74, 111]. The iSSF is a software engineering research and education setting utilized by the top industrial and research collaborators in Europe [74]. Indra Software Labs leads this initiative at the corporate level in Spain in conjunction with Universidad Politécnica de Madrid. The case study was performed within a project to develop a family of metering data management systems in electric power networks for smart grids [8], called Optimeter. This project was developed using Scrum methodology [100]. This project is part of two European ITEA2 projects: IMPONET [55] and NEMO CODED [54].

Smart grids require the ability to manage different resources, from renewal or traditional energy producers to energy consumers. The case study performed on the Optimeter project consisted of the development of a family of software systems that manage meter data from a huge number of these resources. A metering management system captures meter data from telemetering systems or batch processes, loads these data into a database, supports data querying and processing, and provides these data to other systems for billing, forecasting, or purchasing. The customer stated that the high performance of data access and storage was essential to the success of the product. In order to manage meter data with high performance, it was necessary to evaluate several of the large data storage technologies available in the market (see Table 3.1).

The Optimeter project started with a proof-of-concept to evaluate Oracle NoSQL

Table 3.1: Main storage technologies evaluated by the prototypes of Optimeter

Prototype 1	Prototype 2	Prototype 3	Prototype 4
Oracle NoSQL	Hibernate	Hibernate	EHCache
Apache Hadoop	EHCache	Oracle Coherence	Apache Pig
	Oracle 11g	Oracle RAC	Apache Hive
	Oracle RAC		Apache Hadoop

(see Prototype 1 in Table 3.1). Over a one-month period (two sprints, each sprint implied two weeks), the team developed a product that implemented a solution based on the data storage technology Oracle NoSQL. Figure 3.3 shows the architecture that had arisen over that product development. At that time, the architects needed to decide whether or not to invest in flexibility in order to support the variation of the data storage technology, i.e., to support a family of products that support the different variants regarding the data storage technology. This is due to there was a level of uncertainty regarding a future change of the data storage technology to satisfy the restrictions of data accessing performance (see Prototypes 2, 3, and 4 in Table 3.1). In this regard, the architects considered the following expected change scenarios: the possibility of varying one, two, or the three additional data storage technologies, as shown in Table 3.1. At that moment, the customer did not know how many of these candidate data storage technologies had to be evaluated in terms of the performance of massive data loading and querying; nor did the customer know what other business and commercial matters needed to be considered. Due to this uncertainty, it was necessary to obtain information about the ROI for preparing the architecture in order to vary the data storage technology (i.e., the value of flexibility).

This case was selected because the possible options under analysis were well known. In the project, four possibilities of evolution were considered. A more complex case with more complex evolution scenarios would complicate the case study. Due to the main goal of the case study was to study a first contact of the model defined in Section 3.3 in a software project, a non-complex case made easier to understand how the model works. Due to the usage of the model implied many data estimations, a case as the selected, where the possible evolution steps were known, allowed to the development team to provide the required estimations. Therefore, this case study is about expected changes in evolution.

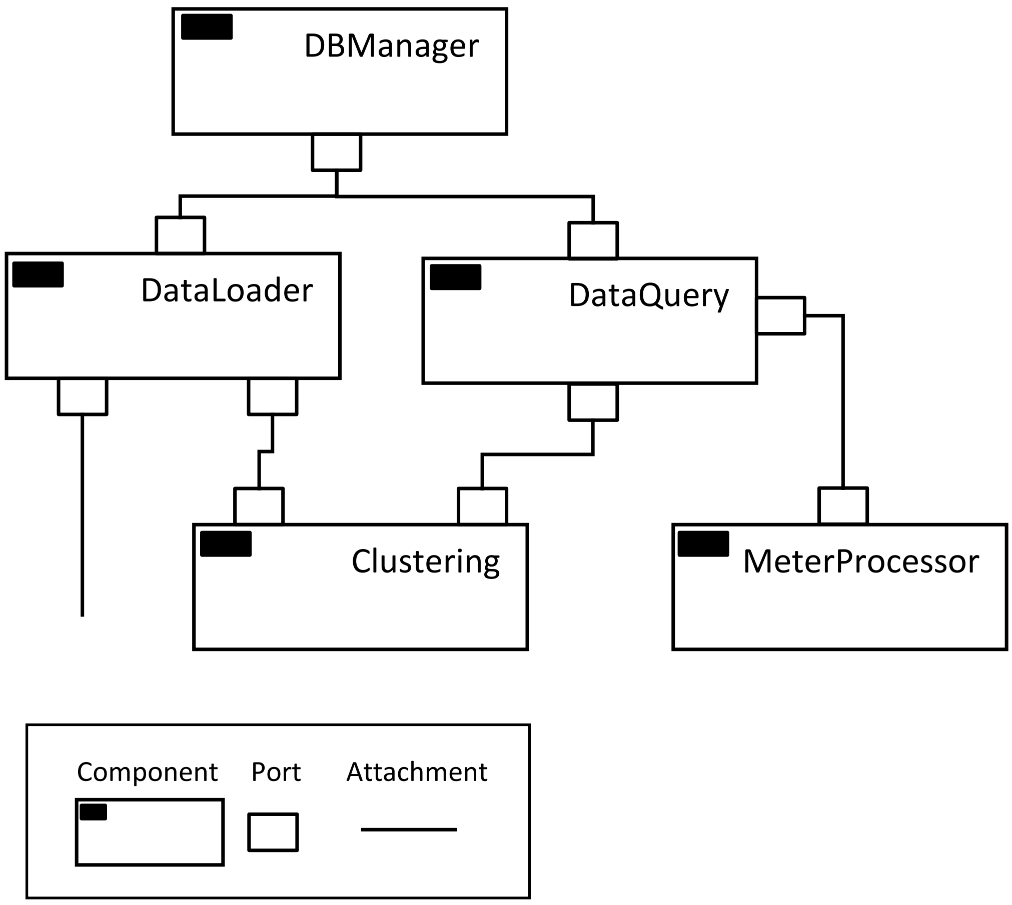


Figure 3.3: Initial software architecture of Optimeter project (month 1)

3.4.4 Theoretical Frame of Reference

The model described in Section 3.3 was used to show how technical debt can be modeled using decision trees. In the case study, it was not only necessary to model technical debt but also to make decisions about flexibility investments. Furthermore, it was necessary to consider the possibility of delaying the decision of investing in flexibility. Therefore there are some changes with regard to the previously defined model. Thus, the first conclusion obtained is that even with a previous model to describe technical debt, it was necessary to adapt it to the specific decisions that had to be analyzed. The main difference is that a new type of node (decision node) is used and that, in the new model, the net value is calculated at the end nodes and the final net expected value is calculated by folding back the decision tree. The following equation specifies how the value is calculated for the different types of nodes.

$$\begin{cases} NV = (V * (C + C_f)) - (CoC * C) - (CoC_f * C_f) - CF & \text{for leaf nodes,} \\ NEV = \sum_{i=1}^{n} NV_i * p_i & \text{for change nodes,} \\ NEV = max(NV_i, \ldots, NV_n) & \text{for decision nodes,} \end{cases} \tag{3.4}$$

where
NV is the net value,
V is the value added by the expected changes if they happen,
C is the number of changes implemented before the investment in flexibility,
CoC is the cost of change before the investment in flexibility,
C_f is the number of changes after the investment in flexibility,
CoC_f is the cost of change after the investment in flexibility,
CF is the cost of investment in flexibility, it will be 0 if there is not investment,
NEV is the net expected value,
p is the probability of changes,
n is the number of outgoing branches from the node.

As an example, Figure 3.4 graphically shows a cost-benefit analysis of one of this types of decisions using a decision tree. The shown scenario is an investment in improving the flexibility of a software architecture to deal with a set of expected changes. Specifically, the software product has to undergo two expected changes over some period of time, and its software architecture should be ready to deal with those changes.

The decision tree method consists of two main steps. The first step involves the construction of a tree that represents (i) the evolution of the software product due to expected changes over some period of time, and (ii) the decisions as to whether or not to invest in preparing the architecture to accommodate the expected changes. Figure 3.4 shows the decision tree over a two-month period. This tree is characterized by three kinds of nodes: decision nodes, change nodes, and end nodes. Decision nodes, which are represented by rectangles in Figure 3.4, are points where a decision on architecture has to be made. Change nodes, which are represented by circles in Figure 3.4, are points where expected changes of the software product could come. Finally, end nodes, which are represented by triangles in Figure 3.4, show the final state of the software product after analyzing all possible scenarios resulting from the expected changes that may or may not occur and decisions that may or may not be made. These nodes make up the branches of decision trees that have an associated probability (e.g., the probability used in this example is 0.33 for all branches, which in this example is assumed that is given by the opinions of experts; see Figure 3.4).

The second step in this process consists of the assessment of the value of changing the software (i.e., the value of investing in flexibility). This investment's net expected value is computed by folding back the decision tree, starting with the end nodes and moving toward the root, using Equation 3.4. This step is composed of two activities. The first activity consists of estimating the net value of end nodes (triangles), that is, the value of the software product after implementing the expected changes of the branch under analysis, minus the cost of implementing those changes, and minus the cost of investing in flexibility, if it is applicable for that node. For example, in Figure 3.4 a scenario is described considering that:

- The value added by an expected change is estimated in $€10,000$.
- The cost of investing in flexibility is $€2,000$.
- The cost of change with a previous investment in flexibility is $€3,000$.
- The cost of a change without a previous investment in flexibility is $€6,000$

In this case the end node marked with A has a net value of $(€10,000 * 2changes) - €2,000 - (€3,000 * 2changes) = €12,000$, while the end node marked with B has a net value of $(€10,000 * 2changes) - (€6,000 * 2changes) = €8,000$. The second activity consists of estimating the net expected value of the decision nodes and the change nodes. The estimation of the decision nodes (rectangles) consists of selecting the branch with the highest net value, while the estimation of the change nodes (circles) is the sum of the net value of the branches weighted by its probability. For example, in Figure 3.4

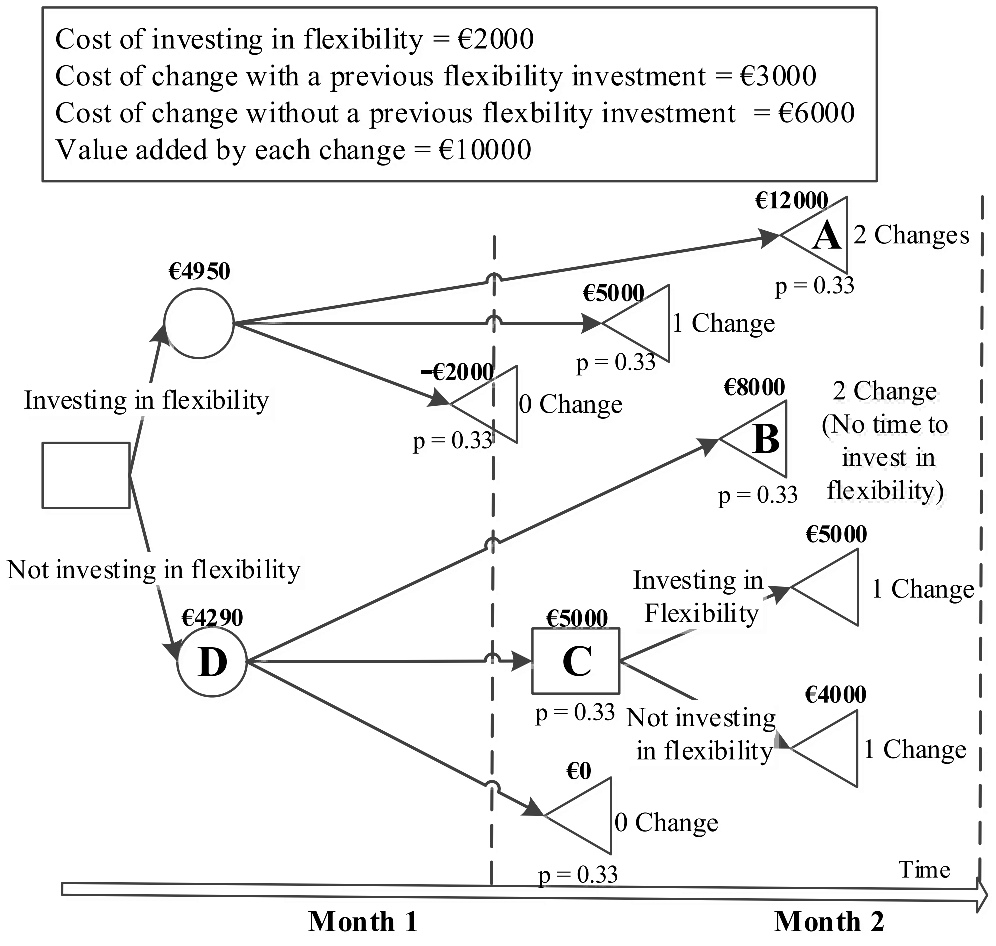

Cost of investing in flexibility = €2000
Cost of change with a previous flexibility investment = €3000
Cost of change without a previous flexbility investment = €6000
Value added by each change = €10000
€12000
A
2 Changes
p = 0.33
€4950
€5000
1 Change
p = 0.33
-€2000
0 Change
p = 0.33
Investing in flexibility
€8000
B
2 Change (No time to invest in flexibility)
p = 0.33
€5000
Investing in Flexibility
1 Change
Not investing in flexibility
€4290
D
€5000
C
p = 0.33
€4000
Not investing in flexibility
1 Change
€0
0 Change
p = 0.33
Time
Month 1
Month 2

Figure 3.4: Example of decision tree used to analyze the option of investing in flexibility in software architecture.

the decision node marked with C has a net expected value of max(€5,000, €4,000) = €5,000 while the change node marked with D has a net expected value of (€8,000 $*$ 0.33) + (€5,000 $*$ 0.33) + (0 $*$ 0.33) = €4,290. This activity ends by estimating the value of the root of the tree (a decision node). This value is used to decide whether or not to invest in flexibility, and when to do so. In this particular example, the value of the root is max(€4,950; €4,290) = €4,950. This means that the best decision is to invest in flexibility from month 1 and not to delay the decision.

The model helps to reason about the possible situations that can occur in the software and that can affect the evolution of the software interest (interest probability). With the definition of the new types of nodes, it is possible to model technical debt as in Section 3.3 and also to consider more complex scenarios about changes in software. The scenario shown in Figure 3.4 represents a project where at month 1 it is not known if the expected change will happen or not. It does not consider the possible value added performing a different activity, for example, adding other features tho the software. However, for a complete analysis, more alternatives could be modeled.

3.4.5 Research Questions

This case study aims to respond to two research questions:

- **RQ1:** Does technical debt concept help to reason about the value of investing in flexibility?
- **RQ2:** What limits could be found in generalizing the propose presented in this chapter evaluating flexibility investments to any other project?

3.4.6 Methods, Data Collection, and Selection of Data

For the purpose of this case study, a process for technical debt management was defined to be integrated into the software development process used in Optimeter project. This section presents the process used to obtain a decision tree model to make decisions regarding flexibility investment in software architecture. The process was performed as a part of the backlog grooming sessions [65]. Backlog grooming sessions give agile teams the opportunity to look further into the future of the product, and can alert them to technical challenges.

The technical debt management process consists of a set of steps to create a model, based on decision trees to estimate when to design for flexibility. Each step is described as follows.

- **Step 1: Analysis of expected changes.** In this step, the architect identifies the expected changes that are to be supported through flexibility. In addition to the expected changes, it is necessary to determine:

  1. The time frame in which the expected changes could happen.
  2. The probability that the expected changes could happen.
  3. The value added by implementing the expected changes.

  This information has to be provided by experts. Depending on the expected changes different actors can be implied in the estimation of the probability of change. If the changes will depend on new contracts, then the responsible of such contracts will be who has the information. If the changes will depend on the adoption of a new technology, then the responsible of such technical decision will be who knows the needed information. Therefore, this probability of change will be subjective. Also, it is possible to use historical data about similar expected changes to estimate how often changes are made in the software.

- **Step 2: Design of alternative architecture solutions to support the expected changes.** In this step, architects propose alternatives with different degrees of flexibility (i.e., different amounts of investment) that have to be analyzed.

- **Step 3: Estimation of technical debt for each one of the alternatives.** The goal of estimating technical debt is to make a cost-benefit analysis of implementing or not implementing the flexibility. To estimate the technical debt the concepts of principal and interest were used.

  In this case study, the principal of a technical debt item, derived from the lack of flexibility, is the cost of improving flexibility. To estimate the principal it is necessary to detect the weaknesses in the system. Static code analysis has been previously used for that task. For example, tools such as SonarQube [107] and PMD [85] can detect architecture anti-patterns in software projects. Knowing the weaknesses, experts can estimate the cost of solving them by applying commonly used cost estimation techniques such as COCOMO or Function Point.

  In this case study, interest of a technical debt item was derived from the lack of flexibility in software. That is, interest of a technical debt item was considered as the additional cost of implementing the expected changes (additional cost of change) if the item was not previously eliminated; i.e., if the software architecture is not flexible enough to introduce the expected changes. The interest of an architecture design solution is the difference between the cost of change of the ideal solution and the cost of change of the alternatives (see "a" and "a + b" in Figure 3.5). However, estimating this interest is not easy because there is not reference for an ideal cost of change; i.e., a solution with zero interest (see ideal solution line in Figure 3.5). In practice, it is possible to use the comparison between the cost of change of the different software architecture alternatives (see "b" in Figure 3.5). This was the approach used in this case study. Finally, Step 2 and Step 3 can be performed iteratively in order to architects can propose architectures that solve the weaknesses detected in Step 3.

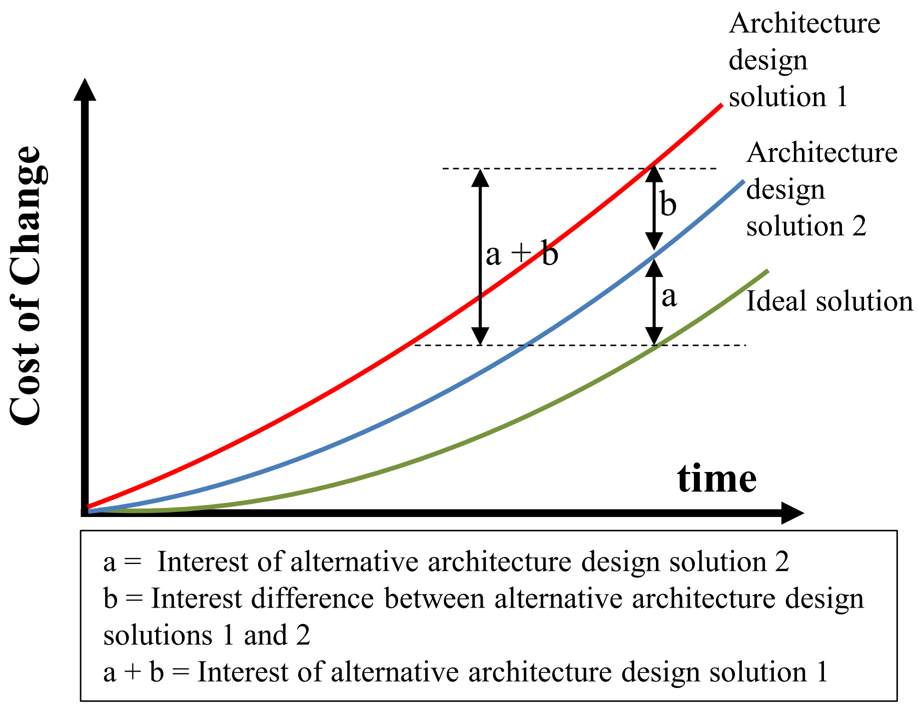

Architecture design solution 1
Architecture design solution 2
Ideal solution
Cost of Change
time
a + b
b
a
a = Interest of alternative architecture design solution 2
b = Interest difference between alternative architecture design solutions 1 and 2
a + b = Interest of alternative architecture design solution 1

Figure 3.5: Technical debt interest

- **Step 4: Construction of the decision tree.** In this step, the architect performs the construction of a tree that represents the evolution of the software system due to expected changes following the instructions described in Section 3.4.4. To do that, the following inputs are used:

  1. The expected changes, their probability, the time frame, and the value that these changes add to the software product under analysis (Step 1).
  2. Alternative design architecture solutions (Step 2).
  3. The constraints imposed by deadlines, resources, and development time needed to implement the expected changes and prepare the architecture to support the expected changes (i.e., designing for flexibility). This means that alternatives that cannot be performed in the time frame, given such constraints, are not considered in the decision tree. For example, in Figure 3.4 an end node marked with B has been modeled to represent a scenario in which sufficient time is not allotted to both invest in flexibility and implement the expected changes.

  The construction of the decision tree requires one to determine the decision nodes, change nodes, and end nodes.

- **Step 5: Assessment of the net expected value of the flexibility investment.** To make this assessment, the following inputs are used:

  1. The estimations of the principal and the interest of each of the alternative architecture design solutions (see Step 3).
  2. The decision tree (see Step 4).

  In this step the model computes the investment's net expected value by folding back the decision tree, starting with the end nodes and moving toward the root (see Section 3.4.4). This computation consists of:

  1. Estimating the net value of the end nodes.
  2. Estimating the net expected value of the decision nodes and the change nodes, until arriving at the root node.

- **Step 6: Make the decision.** Finally, the final decision consists of choosing the investment alternative with the higher net expected value from the root node. The value of the root node is used to decide whether or not to invest in flexibility, and when to do so. In this regard, it is possible that the model suggests delaying the decision until more information is available.

3.4.7 Case Study Protocol

All data estimations were provided by the development team (developers, project architecture, and scrum master). The researchers involved in the case study defined the model in an excel sheet and used the inputs given by the development team to perform the cost-benefit analysis.

3.4.8 Ethical Considerations

For confidentiality reasons, the figures showed in this case study have been altered. They are proportional to the original ones to keep the meaning of decisions and scenarios.

3.5 Case Study Execution

During Optimeter execution, the process described in Section 3.4.6 was applied as follows.

- **Step 1: Analysis of the expected changes.** The expected change in the family of metering management systems under analysis is the variation of the data storage technology (see Table 3.1). Hence, the family of metering management systems will implement three, two, one, or none of the data storage technologies variants. The time frame in which the expected change can happen is three months (the Optimeter project lasted four months, but one month had already been spent on implementing a solution based on the data storage Prototype 1, see Table 3.1). The probability of the expected-change scenarios was estimated on the basis of the customer's knowledge. This knowledge was based on technical, business, and strategic factors, and because they have the last word about which technologies have to be supported. The probabilities can be seen in Figure 3.6. Finally, the value added by implementing each data storage technology variant was assessed by the customers. Therefore, customers estimated the value of each data storage technology that the family was able to support as €60, 000. This value corresponded with the customer's utility of having a family that allows them to vary the data storage technology for different data access performance requirements.

- **Step 2: Design of alternative architecture solutions to support the expected changes.** To provide the software architecture of Figure 3.3 with the

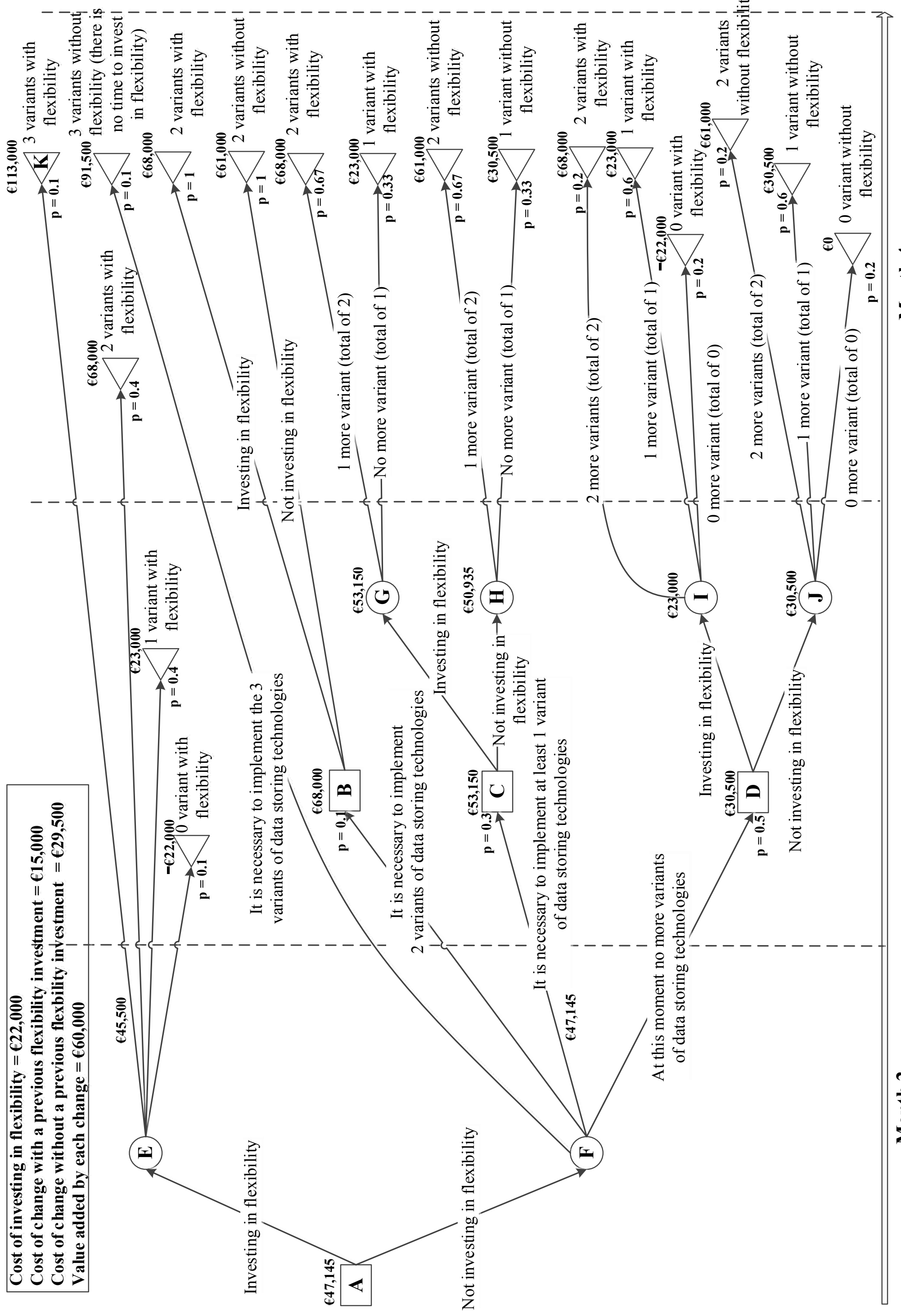

Cost of investing in flexibility = €22,000
Cost of change with a previous flexibility investment = €15,000
Cost of change without a previous flexibility investment = €29,500
Value added by each change = €60,000
A
€47,145
Investing in flexibility
Not investing in flexibility
E
€45,500
F
€47,145
It is necessary to implement the 3 variants of data storing technologies
It is necessary to implement 2 variants of data storing technologies
It is necessary to implement at least 1 variant of data storing technologies
At this moment no more variants of data storing technologies
B
€68,000
p = 0.1
C
€53,150
p = 0.3
D
€30,500
p = 0.5
G
€53,150
H
€50,935
I
€23,000
J
€30,500
Investing in flexibility
Not investing in flexibility
K
€113,000
3 variants with flexibility
€91,500
3 variants without flexibility (there is no time to invest in flexibility)
€68,000
2 variants with flexibility
€23,000
1 variant with flexibility
−€22,000
0 variant with flexibility
€61,000
2 variants without flexibility
€30,500
1 variant without flexibility
€0
0 variant without flexibility
1 more variant (total of 2)
No more variant (total of 1)
2 more variants (total of 2)
1 more variant (total of 1)
0 more variant (total of 0)
p = 0.1
p = 0.4
p = 1
p = 0.67
p = 0.33
p = 0.2
p = 0.6
Month 2
Month 3
Month 4

Figure 3.6: Scenarios analyzed in the case study

flexibility that facilitates the variation of the data storage technology, the architects proposed an alternative design architecture solution (see Figure 3.7). This architecture makes ready the software product to be flexible through two points of variation to support the four data storage technologies shown in Table 3.1. Because a data storage technology is composed of a data manager and a clustering framework, these variation points are defined through:

– Three optional components that implement the data managers
– A Plastic Partial Component [82], i.e., a special type of component that can define variability points inside components, that implements the clustering frameworks.

These components were successfully applied in previous agile projects.

- **Step 3: Estimation of technical debt for each one of the alternatives.** The principal, as defined in Section 3.4.6, is the cost of providing the architecture of Figure 3.3 with the flexibility that facilitates the variation of the data storage technology (i.e., the cost of implementing the variation points shown in Figure 3.7). Additionally, PMD [85] was used to detect current weaknesses (searching anti-patterns that have negative impact on flexibility) giving a total of one GodClass and 10 different coupling problems. The team estimated the effort of implementing the new architecture that finally was valued at €22, 000. However, if no investment in flexibility is made, then this cost is €0. The interest, as defined in Section 3.4.6, is the additional cost of change of supporting a data storage technology variation derived from the lack of flexibility. The cost of change was estimated in terms of the dependencies and the coupling among the components that are affected by the expected changes. Dependencies were obtained from SonarQube [107] and the problems that were detected using PMD and, in terms of effort, they were estimated by the development team that already knew the architecture and they have been working implementing the first prototype the previous month. The cost of change of the architecture of Figure 3.3 was €29, 500 for each data storage technology that was to be supported by the Optimeter family, while the cost of change of the architecture of Figure 3.7 is €15, 000 for each data storage technology. Therefore, the difference in cost of change between the two alternatives was €14, 500 (i.e., the difference between €29, 500 and €15, 000).

- **Step 4: Construction of the decision tree.** The decision tree shown in

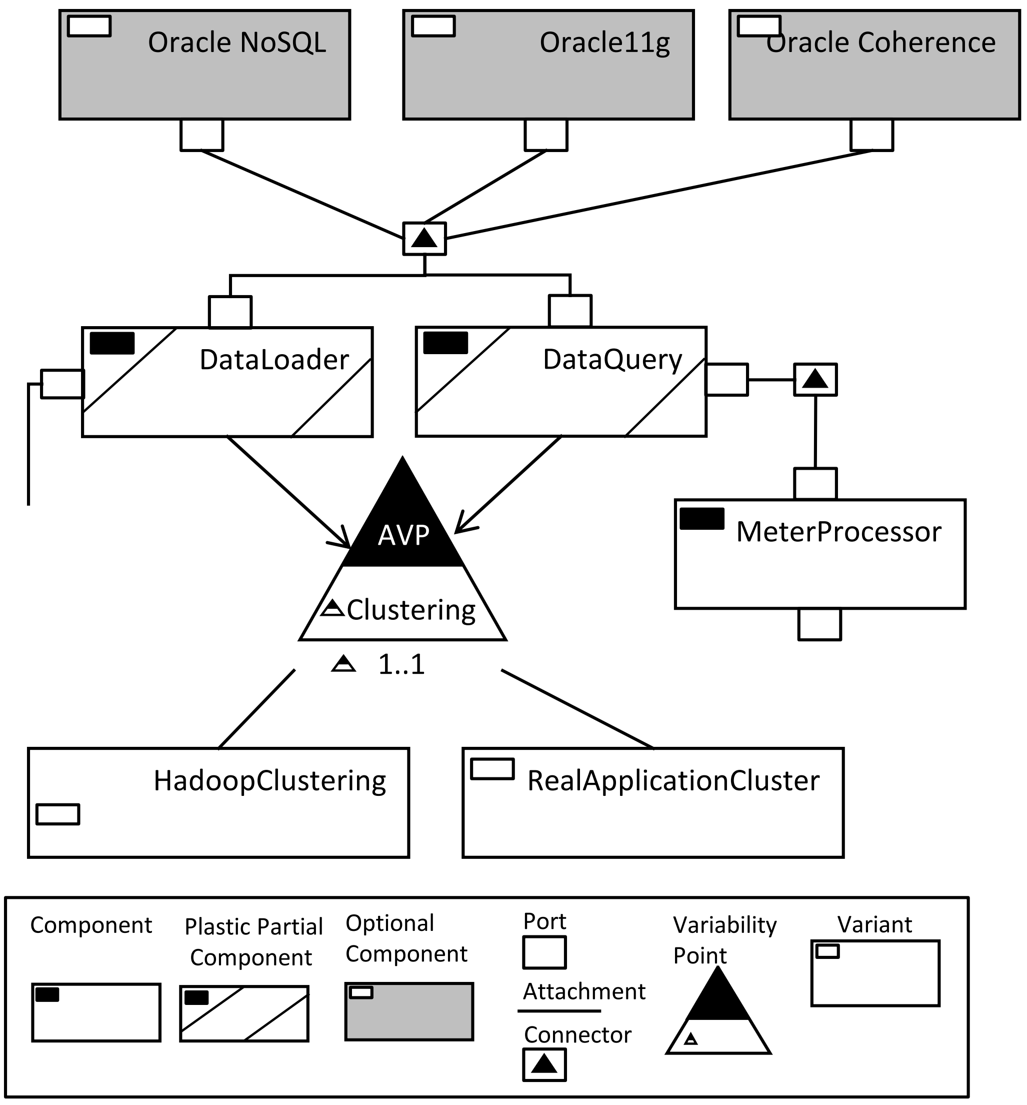


Figure 3.7: Proposed architecture to support the three data storage prototypes in Optimeter project

Table 3.2: Examples of calculus of decision tree nodes using Equation 3.4

Nodes	Net value	Value description
K, end node	€113,000	€113,000 = (€60,000 ∗ 3) − €22,000 − (€15,000 ∗ 3)
B, decision node	€68,000	€68,000 = max(€68,000, €61,000)
G, change node	€53,150	€53,150 = (€68,000 ∗ 0.67) + (€23,000 ∗ 0.33)
A, root node	€47,145	€47,145 = max(€54,500; €47,145)

Figure 3.6 represents the evolution of the Optimeter family due to the expected variation of the data storage technology. While we have shortened the decision tree, it still shows all the relevant information. Concretely, branches that start in node E have been simplified to reduce the size of Figure 3.6. It is necessary to highlight that the three analyzed months are labeled as month 2, month 3, and month 4, because the project started one month before this analysis was performed (see Section 3.4.3). As the tree represents, the decision of investing in flexibility could be made at different moments through the decision nodes. Specifically, it was possible to invest in flexibility in month 2 and month 3 (see nodes A, B, C, and D in Figure 3.6). The tree shows the expected change scenarios in which the three data storage technology variants may be or may be not required (see nodes E, F, G, H, I, and J in Figure 3.6); it also shows the probability that these expected-change scenarios may or may not occur (see the arrowheads of output lines from nodes E, F, G, H, I, and J). Finally, the tree represents the final possible situations as a consequence of the combinations of decisions and changes (see triangles in Figure 3.6). For example, the end node K represents the implementation of the three variations of data storage technology with a probability of 0.1, given the change node E, and as a consequence of deciding on "investing in flexibility" in the decision node A.

- **Step 5: Assessment of the net expected value of the flexibility investment.** In this step, net value is calculated as described in Section 3.4.6. As a result, the decision tree construction has been completed and reflects all of the possibilities that could unfold over time and shows the choices that could be made at each decision node. As example, Table 3.2 details some results of nodes of Figure 3.6.

- **Step 6: Make the decision.** The final decision was made on the basis of the expected net value obtained for the analyzed scenarios (see decision node A in Figure 3.6). The decision tree indicates that it is better to not invest in flexibility in month 2 because the value of node marked with F is higher than the value of node E; and therefore is more profitable to delay the investment because the uncertainty could decrease in month 3 (see Figure 3.6). This result is due to the value gained from waiting for more information about the project. That is, waiting to resolve uncertainties before deciding whether or not to invest.

3.6 Findings

In this section, the results for each research question formulated in Section 3.4.5 are described.

3.6.1 RQ1: Does technical debt concept help to reason about the value of investing in flexibility?

Technical debt was useful to analyze the cost and benefit of flexibility investments. It helped to reason about the impact of future changes. However, the model used in this case study might present some challenges to be used in a bigger project where estimations were more difficult to be obtained.

One of the challenges is derived from the fact that the case study was focused on expected changes. This simplified the scenario definition and the production of the estimations. To define architectures for specific expected changes is easier than to be ready for changes in general. Furthermore, the value of the investment can be achieved because knowing the expected change it is possible to reason about its expected value. Therefore, this approach presents limits to be used to assess the value of flexibility in general, considering non-expected changes. However, the technical debt concept helped to reason about the cost-benefit analysis considering long-term impact of possible changes in the future. Thus a deeper study on technical debt will be valuable to analyze how to use it in more general evolution scenario in software development. It has special interest to study how to use technical debt with fewer sources of information to make estimations. Lack of information is a more real scenario in software development than a scenario with perfect information.

Additionally, technical debt management was integrated into the software development process of the project. In this direction, to obtain the input data from models (principal, interest, and probabilities) in a systematic/automatic way would be a landmark. This automation is especially necessary in industrial projects in order not to disturb the normal project development. Another challenge is extending the model with more factors, for example, considering alternative developments instead of paying the principal.

As the model uses estimated data as input, it will model realistically the technical debt of the system only if the inputs are correct. Therefore, it is important to obtain good estimations of such inputs. Decisions are highly influenced by the probability of the possible evolution paths. These evolution paths affect the interest probability. Consequently, methods to estimate interest and interest probability with precision are required, especially for scenarios where developers, or other stakeholders, can not provide those estimations.

3.6.2 RQ2: What limits could be found in generalizing the propose presented in this chapter evaluating flexibility investments to any other project?

As said before, the proposed model is based on estimations; the more precise are estimations, the more precise will be the conclusions from reasoning. The estimations needed to use the presented model might be difficult to be obtained in big software projects where many times the knowledge of the project is dispersed, when not lost, and the size and complexity of software make difficult these estimations. The estimations used in the case study were mainly subjective. But it should be highlighted that they were intended to support qualitative reasoning, not quantitative; therefore they should be simply shown the right trend. The project analyzed in the present case study was a small project and the team was already working on it. Therefore, they had all the needed knowledge and the expertise to make subjective estimations.

The scenario was simplified by not considering the value of possible alternative actions that could be implemented by the development team. For a complete analysis, alternatives in which to spend the effort if it is not done an investment in flexibility should be considered. However, more information would have required more complex scenarios that might have complicated the analysis.

Decision trees were found useful in modeling change scenarios. However, even in a framed scenario like the presented in this case study, the decision tree expanded quickly. This indicates that this technique, even useful to reason about specific scenarios, could not be suitable for more complex scenarios where more alternatives, variables, and not expected changes could be evaluated.

Another simplification was done by estimating the value added of each new technology with the same value. The same was done with the cost of implementing the changes. In the present study, the changes were very similar. They consisted in implementing prototypes with the same functionality but with technical differences. In other projects, different changes might imply different cost and benefit. Therefore, these differences in cost and benefit should be considered.

Finally, the technical debt model defined in Section 3.3 had to be adapted to the specific type of decisions that had to be made in the project. Therefore, it seems that depending on the type of decisions to be made, different sources of information could be required. Then, it could be valuable to research what sources of information and what considerations should be taken into account to manage technical debt.

3.7 Limitations

To improve the internal validity of the results presented, the independent variables that could influence this case study have been identified as follows: the architects' experience, the influence of the project's size, the architecture's complexity, and finally the complexity of the possible expected-change scenarios, which cannot be reduced due to the inherent nature of case studies, which normally focus on one project. In particular, we have reduced the complexity of scenarios to guarantee the understandability of the case study by using the same value to estimate the value added by each data storage technology. However, if we had estimated the value added by each data storage technology with different values, the expected-change scenarios would have been more complex, and consequently, the scenario would be more difficult to be understood. Finally, it is important to emphasize that if the expected changes are not well-scoped, then it would be difficult to estimate the cost of change. However, the major limitation in the case study research concerns external validity because only one case has been studied. In return, case studies allow one to evaluate a phenomenon, a model, or a process in a real setting. This is important in software engineering in which a multitude of external factors may affect the validation of results, and where other techniques, such as formal experiments, are not considered to be conducted in controlled settings, even though formal experiments permit replication and generalization.

3.8 Conclusions

In this chapter, the concepts to add value from the internal quality of software identified in Chapter 2 were used in a case study to obtain deeper knowledge about them. To achieve this goal a model to make cost-benefit analysis based on technical debt was defined. The model was used in a research project in the development of software prototypes. The model was useful by providing a solution for the flexibility investment decision making.

Delay decisions can have value in contexts where there is uncertainty. This situation corresponds with scenarios where the expected benefit of investing today in flexibility does not compensate the risk that the expected changes do not happen in the future.

The time frame used in the analysis is very important. Depending on the time frame different decisions could be made. Therefore, it is necessary to study how to choose the time frame for the analysis. For example, the expected date for retiring the software or the time-to-market of the next release could be used.

Technical debt concept, through principal, interest, and interest probability, helps to reason about the value of changes in the internal quality of software. Technical debt links improvements in the internal quality of software with the value that could be added to the software in the future. This confirms the conclusions of Chapter 2. Therefore, technical debt should be considered for decision making with a value-based software engineering perspective. Thus, a deeper study on technical debt will be valuable to analyze how to use it in more general evolution scenario in software development. It has special interest to study how to estimate technical debt in a context with fewer sources of information, because lack of information is more real scenario in software development than a scenario with perfect information. It is also necessary to research what sources of information and what considerations should be taken into account to manage technical debt.

The challenges that have been described in Section 3.6 have been identified by using the concept of technical debt in a small project. Therefore, it is required to test the same concepts in big projects. Therefore, it is necessary to identify what techniques could be used in big projects to obtain the estimations that are required to make decision based on technical debt.

Part III

Identification and definition of the elements that are required to create models that help make decisions in software evolution

Chapter 4

A Framework for Technical Debt Management

The goal of this chapter is to identify the entire set of elements that have been used to manage technical debt. Elements are understood as the concepts used to implement technical debt management, regardless of their nature. Without a minimum knowledge of these elements, it would not be possible to define the models or techniques required for general technical debt management. This chapter is an excerpt from the following articles:

- Carlos Fernández-Sánchez, Juan Garbajosa, Agustín Yagüe, Jennifer Pérez, Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study, Journal of Systems and Software, Volume 124, February 2017, Pages 22-38, ISSN 0164-1212. [39].

  Copyright ©2016 Elsevier Inc.
- C. Fernández-Sánchez, J. Garbajosa, and A. Yagüe, A framework to aid in decision making for technical debt management, IEEE 7th International Workshop on Managing Technical Debt (MTD) in conjunction with 31st International Conference on Software Maintenance and Evolution, Bremen, 2015, pp. 69-76. [37].

  Copyright ©2015 IEEE.

4.1 Introduction

The second contribution of this thesis, Contribution 2, is *Identification and definition of the elements that are required to create models that help make decisions in software evolution.* Previous chapters of this thesis highlight that technical debt links improvements in the internal quality of software with the value that could be added to the software in the future. Therefore, the technical debt concept was identified as a key concept for applying value-based software engineering principles to software evolution. It was also indicated that it is necessary to research what sources of information and what considerations should be taken into account to manage technical debt. To get a deeper understanding of how to manage technical debt a systematic mapping study in technical debt management. The goal is to define a theoretical framework for technical debt management. The framework will be defined by the elements necessary for managing technical debt, as well as, by the stakeholders' possible points of view. The present chapter shows how the systematic mapping was performed and the outcomes that were obtained.

The remainder of the chapter is organized as follows: Section 4.2 will discuss previous related studies. Section 4.3 will present the methodology and the processes used in this chapter. Section 4.4 will describe the identified elements. Section 4.5 will present the mapping of the elements according to the stakeholders' points of view. Section 4.6 will present a retrospective of the findings presented in this study. Finally, threats to validity, conclusions, and recommendations for future work will be presented in Section 4.7 and Section 4.8, respectively.

4.2 Related Work

There are several previous reviews of the literature on technical debt. Tom et al. [114] performed a multivocal literature review to create a taxonomy of the phenomenon of technical debt. Alves et al. [5] performed a systematic mapping to create an ontology of technical debt. Li et al. [69] performed a systematic mapping identifying activities to perform in technical debt management. Alves et al. [6] performed a systematic mapping to identify types of technical debt and methods used for technical debt management. Finally, Ampatzoglou et al. [9] performed a systematic literature review to study the financial aspects of technical debt. In the literature, several studies have addressed the phenomenon of technical debt [114, 5, 6, 9], the methods used for technical debt management [6], and the activities used in performing in technical debt management [69]. The goal of this chapter is different from these previous studies since we are specifically

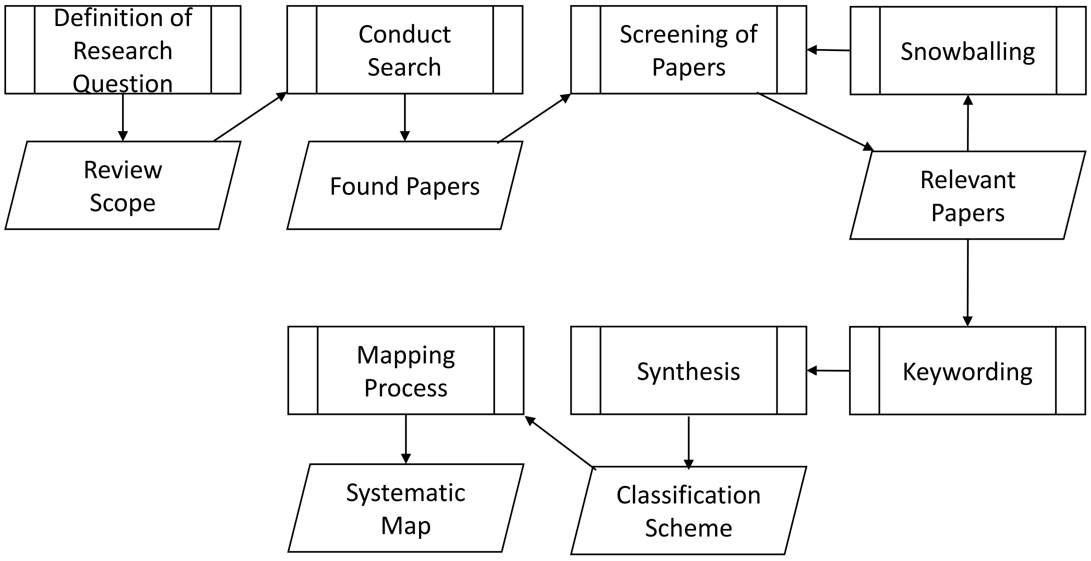


Figure 4.1: Systematic mapping process (adapted from [84])

interested in the elements required to manage technical debt. Therefore it was necessary to perform a new literature review. Some specific differences between the findings of our study and what can be found in the literature are addressed in Section 4.6.

Finally, Falessi et al. [M17] identified the requirements for the tools utilized to manage technical debt. We have extended their work by incorporating the ways in which other authors have considered these requirements.

4.3 Methodology and Research Process

To identify the elements of technical debt management that have been addressed in the current literature, this study utilized systematic mapping. The systematic mapping was performed by following Petersen et al.'s guide [84]. Systematic mapping studies are designed to provide a broad overview of a research area and are consequently appropriate for the goal of this research, which is, to identify the elements that must be taken into account to manage technical debt efficiently. All the steps in the process are shown in Figure 4.1 and described in the following subsections.

4.3.1 Research Questions

RQ1: What elements have to be considered when making decisions concerning technical debt management in software projects?

RQ2: What elements are considered from the various stakeholders' points of view?

To answer these research questions, this study focuses on publications about technical debt management.

4.3.2 Conduct Search

The research was focused on methods, techniques, and suggestions for technical debt management. After some trials with the strings "debt" and "technical debt", the second one was select because it returned all the previously known papers, and the string "debt" led to too many false positives. However, to reduce the risk of leaving out relevant papers, the search was complemented with the snowballing technique (see Section 4.3.4). Therefore, an automatic search method using the term ***"technical debt"*** was used to search for papers about technical debt in the following digital repositories: IEEE Xplore, ACM, Scopus, ScienceDirect, Web of Science, and SpringerLink. *Full-text* search was used when this option was available (IEEE Xplore and SpringerLink), as well as for searching in metadata in other cases (ACM, Scopus, ScienceDirect, and Web of Science). The search included all the papers published until and including 2015. The total number of articles obtained (including duplicates) was 971.

4.3.3 Screening of Papers

The selection process consisted of two levels. In the first one, papers that provided enough information in their title and abstract to be excluded were eliminated. In the second one, the full text of the papers was analyzed. To select the articles, the following inclusion and exclusion criteria were used:

- To be included, the paper had to describe parts, activities, tasks, elements, or considerations for technical debt management.
- To be included, the paper had to be published in a journal, conference proceedings, or workshop proceedings. Only book chapters referenced by another included study were included.
- Papers that failed to address technical debt management in detail were excluded.
- Papers published as abstracts, call for workshops, tutorials, talks, or seminars were excluded.

At the end of this step, the number of selected relevant papers was 61.

4.3.4 Snowballing

In order to include all relevant papers, the bibliography of each included paper was screened by using the snowballing technique. Snowballing refers to using the reference list of a paper to identify additional papers [117]. This technique is commonly used in systematic mapping studies [117, 69, 81]. Each new identified relevant paper starts a new iteration in the snowballing process. When no more relevant papers are found, the process ends.

At the end of this step the number of selected relevant papers was 63, that is, two papers were added. The two papers were identified in the first iteration of the snowballing process. The second iteration did not lead to more articles, and consequently, the snowballing process ended. The list of the selected papers is included Annex C.

4.3.5 Keywording

The process used to identify the elements required for technical debt management was adapted from the "keywording" described by Petersen et al. [84]. Figure 4.2 shows the steps used in this process. The main difference from [84] is that in the present study, the process started using the *full text* of the selected papers instead of the abstracts. The reason for this decision was that the entire description of the technical debt management activity was not usually present in the abstracts of the papers. The process was iterative. After the first classification was obtained, it was refined by contrasting the various elements found. The classification scheme then was updated by binding similar elements. This additional step was necessary because it was found that the same concept (or very similar concepts) was used in different contexts or with different names. The criteria used to extract the elements corresponded to the identification of requirements, estimations, analysis, and activities (including inputs and outputs) in technical debt management that were used or suggested in the selected papers.

4.3.6 Synthesis

After the keywording step, a synthesis step was performed to refine the classification scheme previously obtained. In this synthesis, the reasons for and intentions to use each element were also extracted from the original papers. A method analogous to the constant comparison used in qualitative data analysis [108] was used to find commonalities in the intentions of using each element. Following this process, the definitions of the elements were created. Additionally, three types of elements were identified. The

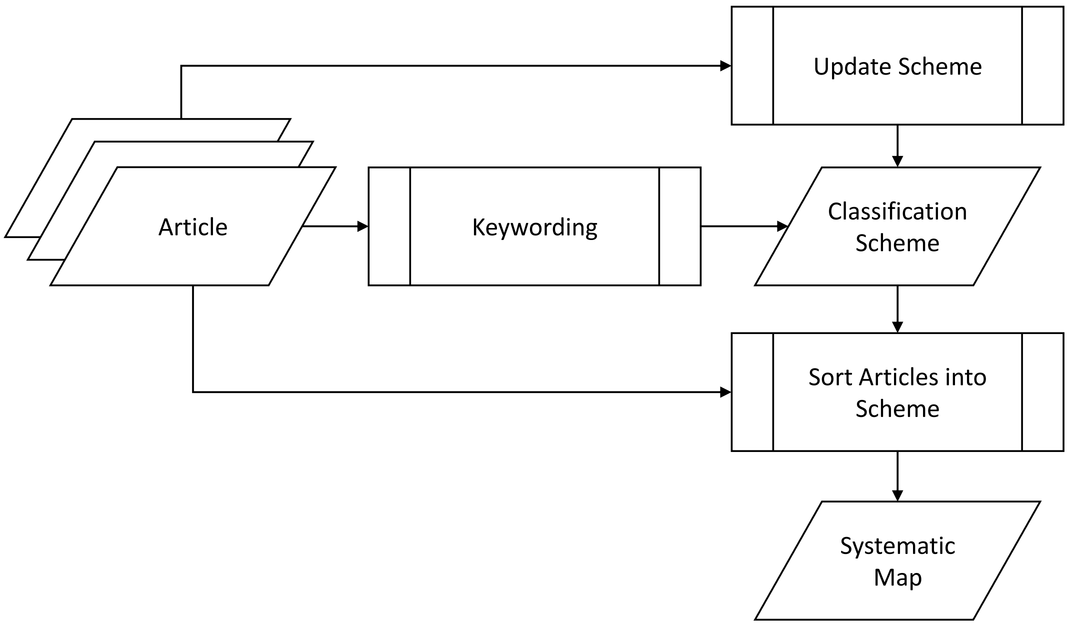


Figure 4.2: Keywording process (adapted from [84])

types of elements defined are explained in Section 4.4.2. As discussed in Section 4.6, this synthesis provided a schema that in practice creates a framework for the elements for technical debt decision making.

4.3.7 Mapping Process

The papers included in the current review were classified using the various elements found (see Section 4.4) and the points of view (see Section 4.5). It was found that each paper included several elements and/or points of view.

4.4 Elements of Technical Debt Management

This section addresses the first research question *What elements have to be considered when making decisions concerning technical debt management in software projects?* Table 4.1 shows the sources that either used or suggested the usage of specific elements for technical debt management. The elements were identified following the steps described in Section 4.3.5, and they were defined by following the steps defined in Section 4.3.6. Section 4.4.1 includes a detailed explanation of the identified elements and Section 4.4.2 shows a categorization of the elements.

4.4.1 Elements

In this section, the elements are described.

4.4.1.1 E1 Technical debt items

To manage technical debt properly, it is necessary to know the sources that originated technical debt in the system. This element is basic because if it is not known that a problem exists, it is not possible to manage it. In fact, to identify technical debt items is usually the first step in managing technical debt [M12, M53].

The identification of technical debt items can include establishing a list of the bad practices that create debt [M30, M36] and identifying the potential kinds of technical debt from the sources of technical debt [M17], as well as determining the part of the system that must be refactored [M8, M28]. Therefore, there are many potential sources of technical debt at any time in any system [M6].

4.4.1.2 E2 Principal

In decision-making, it is essential to know the cost that is required to remove a technical debt item by changing the software. Many authors use the term principal to refer to this cost. The principal of a technical debt item is the cost to be paid to eliminate the item [114]. Estimating the principal was a basis of technical debt management according to most authors (see Table 4.1). It is possible to estimate the principal as a function of three variables: the “should-fix” items with violations, the hours needed to fix each violation, and the cost of labor [M12].

An important consideration is that technical debt is *context* dependent [M54]. Therefore, solving a weakness can cost more or less depending on the project or even the subsystem within a project [M54]. However, because estimating the principal in terms of single values is difficult, practitioners seem to think in ranges of values or best-case, worst-case, and most probable scenarios, rather than single values [M17, M8].

4.4.1.3 E3 Interest

In the decision-making process, to have complete information about technical debt, it is necessary to know not only the principal of the technical debt items but also the cost of not removing them. This cost is usually termed the interest. The interest is the cost to be paid over time if a technical debt item is not eliminated. This cost can include the extra cost of modifying a component that needs refactoring compared to the cost of modifying it after refactoring [M6]. Similar to the principal, estimating the interest was deemed fundamental by most authors (see Table 4.1). The interest can be seen in various ways. It is possible that depending on the project and its context, the interest could be non-linear, or it could have limits, maximums, or minimums [M17]. Similar to the principal, an important consideration is that technical debt is *context dependent* [M54]. Hence, the same detected weakness can imply more or fewer future costs depending on the project or even the subsystem within the project [M54].

4.4.1.4 E4 Interest probability

The interest has a probability of being paid [M53, M17, M6]. That is, it is necessary to know such probability in order to have a real estimation of the interest. This uncertainty exists because the interest must only be paid under some scenarios [M29]. The probability of paying the interest will depend on the probability of the occurrence of future events [M17, M33, M50]. For example, if a technical debt item implies that more effort is required in maintenance activities but that the item will not be changed over time, the interest does not need to be paid. Hence, in this example, the probability of changing this item will parallel the probability of interest. Therefore, the interest is determined by a set of relevant change scenarios [M33], that is, the probable changes that will occur in the future. These scenarios will include, for example, the addition of new functionality, changes in the non-functional requirements, the solutions to problems or bugs in the system.

4.4.1.5 E5 Technical debt impact

The complete technical debt estimation consists of principal and interest estimations, including the interest probability. To manage technical debt, it is necessary to use models in which items are prioritized based on their interests and principals [M53]. The aim of ranking technical debt items is to identify which items should be resolved first, depending on the business's goals and preferences [M33]. Decisions about technical debt should be made in terms of cost-benefit analysis [M53, M17]. Thus, cost-benefit estimations are necessary to make decisions about removing technical debt [M8]. The *business value* of conducting any activity lies in the difference between the cash flow stream of performing the activity and the cash flow stream of not performing it [113]. That is, a technical debt item must be removed when doing so is profitable. The basis of cost-benefit analysis is to identify the items with the highest amount of technical debt and consider the cost of fixing them [M53]. Cost-benefit analysis usually includes several variables. The most obvious are the principal (the cost paid to remove a technical debt item) and the interest (the cost to be avoided by removing a technical debt item).

Additional costs items could be considered, such as time-to-market penalizations and quality issues. Benefits, such as quality improvements and customer satisfaction could also be considered. Additional information that could be used includes the number of features that a software release contains, the time required to deliver the release, and the technical debt that is generated because of forced quick development [M46].

The outcome of technical debt management should be reviewed in terms of its economic consequences [M17] by taking into account business considerations [M7]. That is, technical debt must be quantified [M45]. It is not always easy to express technical debt using economic data. However, without expressing the economic consequences, it is difficult to quantify the effects of executing or not executing a decision. Tom et al. [114] classified technical debt costs into four types: morale, productivity, quality, and risk [114]. However, one problem is that the financial effects of technical debt are not always direct [114]. For example, technical debt can cause low morale in the development team resulting in systemic problems, such as developer turnover [114, 83]. Another issue is that it is necessary to balance rigor with the usability of the estimation method. A very complex and rigorous mathematical estimation model could be highly precise in its prediction and yield highly reliable results, but it could be unusable in practice because it is overly complex when it is adapted to real, large projects [M17]. Finally, some special situations must be considered in cost-benefit analysis. For example, when a system is "retired", its technical debt is removed [114]. Another issue is that organizations probably do not have adequate resources to fix all the identified technical debt items [M30]. In summary, a complete cost-benefit analysis must take into account all factors. That is, it must consider not only the principal and interest but also the project constraints, including the deadline, budget, and effects [M30, M33]. Because of this complexity, techniques that help visualize the estimated effects of the technical debt (not technical debt itself) could greatly help decision-making.

4.4.1.6 E6 Automated means

Big projects can generate high volumes of data. Therefore, because of feasibility, it is necessary to obtain estimates automatically to manage technical debt which avoids the negative effect of intrusiveness in the normal development process [M45]. However, the collection of measures is an extra step for developers, who are already overloaded, and might compromise the success of technical debt management [M54]. Hence, the source code, a project's revision history, a project's bug history, and other similar data sources can be mined using automatic tools to obtain the information required to estimate technical debt automatically [M53, M17] and to propose potential items for refactoring [M8].

4.4.1.7 E7 Expert opinion

Expert opinions about the system are required to manage technical debt because they provide knowledge that cannot be obtained from available software information. Together with automated estimates to manage technical debt, the opinions of the people that know the system deeply are required [M53, M17]. This information can include new contracts to be signed, expected changes to be made, or new technologies to be adopted [M20], helping the manager to provide and apply information regarding issues such as uncertainty about the measures and judgments, the system's external context, and the knowledge of experts (project managers, architects, etc.) [M8]. Finally, the goal of technical debt management methods and tools is to provide the necessary information to human decision-makers [M54].

4.4.1.8 E8 Scenario analysis

In decision-making, it is necessary to estimate the consequences of the decisions made about the system. Therefore, managing technical debt includes defining and analyzing multiple potential scenarios [M17]. By analyzing scenarios, managers can acquire information about the effects of the technical debt if certain events occur in the future, which is discussed in Section 4.4.1.4. Various possible implementations can be analyzed to determine which one is the best. In technical debt management, the goals of scenario analysis are as follows.

- To set targets for debt and specify the level that is acceptable for the project or organization [M30].
- To identify the effects of non-fixed technical debt issues on multiple releases. This is performed by using change scenarios [M32, M33, M20]. Change scenarios represent the probable future changes that the system will accommodate, and they are used to determine the amount of interest to be paid because of technical debt [M50].
- To identify when it is profitable either to implement new functionality early (taking more technical debt because of the quick release or when it is profitable to release functionality slowly but with less technical debt [M46]. The scenarios to be analyzed can include the various paths followed to deliver features in the release

planning [M42], that is, to analyze how much effort to invest in either refactoring, architecture design, or the addition of new features [M6] by researching various release scenarios [M25].

- With this information, the manager will be able to choose which decisions to implement with the highest probability, and he or she will know the other possible alternatives and the system's technical debt evolutions. The output of the technical debt management must be in the form of possible scenarios as well as the probability of their economic consequences [M17].
- It is necessary to sketch and assess potential alternatives to the benefits and costs to support choosing the most appropriate for handling the technical debt [M7]. This process includes testing various possible scenarios to analyze the effects of removing some technical debt items, that is, performing a "what-if" analysis [M8, M54].

4.4.1.9 E9 Time-to-market

One important constraint to consider is the time-to-market [M17]. This element includes the resources and constraints involved in achieving a project goal on time. The solutions to be implemented could be useless if they cannot be implemented within a certain time. In some situations, it could be necessary to release a product by the deadline without all its expected characteristics. In certain environments, being the first to market is vital to obtain customers. In this situation, long-term software problems are not important because they are not visible to the product's customers, and these problems are not very important to the software company. In the long term, the problems will be relevant only if the product or the sponsors obtain customers in the short term [114]. To consider fully the costs and benefits of incurring technical debt, it is necessary to take into account the *release planning* of the product under analysis [M42]. In summary, a tradeoff between release characteristics and technical debt must be made to manage technical debt [M46].

4.4.1.10 E10 When to implement decisions

Managing technical debt requires making the decision either to pay the principal of technical debt items or to continue to pay the interest on such items. This element is greatly influenced by the constraints on and the availability of the development team's resources. It is necessary to know when to implement this decision [M53, M17]. Some decisions may include when to implement a feature of a product, or when to refactor some part of the system to improve some of its qualities [M42]. Hence, such decisions affect the release dates and planning [M25].

4.4.1.11 E11 Technical debt evolution

To know how technical debt affects a system it is not enough to have a snapshot of the technical debt in the system. In order to contrast the technical debt of the system with changes in the context, it is necessary to know how the technical debt evolves in the system. Hence, it is necessary to track technical debt over time [M45]. This tracking requires methods to determine the level of technical debt and its evolution [M6]. Monitoring technical debt consists of keeping track of changes in the costs and benefits of unresolved technical debt items [M33]. By monitoring technical debt frequently, it is possible to react quickly [M30]. Thus, monitoring the evolution of the economic consequences of technical debt [M17] is important. To determine how the system will perform in the future, it is necessary to consider the time-frame required for such an analysis [M20]. Because technical debt implies a cost over time, the time-frame will structure the analysis and enable a cost-benefit analysis. One threshold for the time-frame could be the date estimated for retiring the software [M46]. Other potential time-frames could be obtained from the project release plan [M23] or from the project roadmap.

4.4.1.12 E12 Technical debt visualization

Managing technical debt without the visibility of the technical debt items, and the software artifacts (files, modules, packages, etc.) in which the items are accumulated is not possible. Therefore, having the means to see how technical debt affects the system or the development process is highly recommended [M4]. Defining a visual language for the entire organization allows for fast and transparent communication between people and entities [M30]. Hence, it is possible to determine the relative effects of technical debt with regard to other activities [M45]. However, the visualization technique must have the ability to summarize the information required for high-level analysis, and it must include the possibility of analyzing technical debt at lower levels, such as at the subsystem or component level [M54]. The visualization of technical debt is especially important in architectural technical debt [M5]. This kind of technical debt usually implies several source code artifacts, such as classes and configuration files. Using mechanisms to determine how these artifacts are grouped helps to clarify the distribution of technical debt throughout the system.

4.4.2 Grouping of elements according to their use in technical debt management

Figure 4.3 shows the grouping of the identified elements according to their use in technical debt management, which were found in the literature after applying the synthesis step describe in Section 4.3.6. In practice, a **taxonomy** of the elements has been obtained. The analysis showed that the elements could be classified into three main groups: basic decision-making factors, cost estimation techniques, and practices and techniques for decision-making. These are explained in the following paragraphs. *T1 basic decision-making factors* are elements that represent the necessary information about the system technical debt. That is, these elements are information that it is needed to make decisions about managing technical debt. Therefore, these elements are mainly focused on the identification and measurement of technical debt. *T2 cost estimation techniques* are elements focused on how a technical debt management model should be implemented. The main difference from the basic decision-making factors is based on the degree of human intervention. Therefore, models can be implemented using automatic tools, manual processes or a mix of both. Finally, *T3 practices and techniques for decision-making* are elements focused on the considerations that must be taken into account, in addition to the technical debt estimations, to manage technical debt. Elements of this type indicate requirements to be taken into account in addition to the basic decision-making factors. These elements draw attention to the fact that it is not enough to simply identify and measure technical debt. It is also necessary to integrate technical debt management into the project management process as one of its activities.

4.5 Technical Debt Management Elements from the Stakeholders' Points of View

In this section, the second research question is addressed: *What elements are considered from the various stakeholders' points of view?* Although the findings showed that various stakeholders are involved in technical debt management, the most appropriate classification system to use was not evident. Clements et al. proposed, in [23], three stakeholders' points of view, namely (software) engineering, technical management, and organizational management, which were used in the first version of the work. Recently, in SWEBOK V3 [16], the terms engineering management, and business organizational were used. This usage reflects an evolution in the understanding of how software prod-

Table 4.1: Elements of technical debt management identified in the literature. Every element identified is included in the column *Elements*. The references in which it was identified are shown in the column *Sources*. The number of references in which it was identified is shown in the column *Count*. The column on the left classifies the elements according to their use in technical debt management (see Section 4.4.2).

	Elements	Sources	Count
T1 Basic decision-making factors	E1 Technical debt items	[M6] [M8] [M30] [M17] [M11] [M12] [M24] [M23] [M22] [M27] [M29] [M31] [M33] [M34] [M35] [M36] [M41] [M43] [M47] [M48] [M49] [M50] [M51] [M52] [M53] [M54] [M55] [M57] [M59] [M60] [M62] [M63] [M28] [M32] [M9] [M26] [M19] [M61] [M14] [M44] [M40] [M15] [M1] [M16] [M18] [M39]	46
	E2 Principal	[M6] [M8] [M17] [M3] [M11] [M12] [M20] [M13] [M24] [M23] [M22] [M27] [M33] [M38] [M41] [M43] [M50] [M51] [M53] [M54] [M56] [M62] [M63] [M28] [M32] [M9] [M26] [M19] [M61] [M44] [M40] [M15] [M1] [M16] [M18]	35
	E3 Interest	[M6] [M8] [M17] [M3] [M11] [M12] [M20] [M13] [M24] [M23] [M22] [M27] [M33] [M41] [M43] [M50] [M51] [M53] [M54] [M56] [M62] [M63] [M28] [M32] [M9] [M26] [M19] [M61] [M44] [M40] [M15] [M1] [M16] [M18]	34
	E4 Interest probability	[M6] [M8] [M17] [M20] [M24] [M23] [M22] [M27] [M33] [M50] [M51] [M53] [M54] [M19] [M14] [M44] [M40] [M1]	18
	E5 Technical debt impact	[M6] [M7] [M45] [M8] [M30] [M10] [M17] [M2] [M3] [M11] [M12] [M20] [M13] [M24] [M23] [M22] [M27] [M31] [M33] [M35] [M36] [M37] [M41] [M42] [M43] [M46] [M47] [M48] [M49] [M51] [M53] [M54] [M57] [M62] [M28] [M58] [M32] [M9] [M19] [M61] [M44] [M40] [M15] [M1] [M16]	45
T2 Cost estimation techniques	E6 Automated means	[M45] [M8] [M30] [M17] [M12] [M31] [M36] [M38] [M47] [M49] [M53] [M54] [M55] [M57] [M59] [M28] [M58] [M9] [M26] [M19] [M14] [M40] [M15] [M16] [M18] [M39]	26
	E7 Expert opinion	[M8] [M17] [M20] [M22] [M49] [M51] [M53] [M54] [M28] [M32] [M26] [M14] [M44] [M40] [M15] [M1]	16
T3 Practices and techniques for decision-making	E8 Scenario analysis	[M6] [M7] [M8] [M30] [M17] [M3] [M20] [M25] [M27] [M31] [M33] [M42] [M46] [M47] [M50] [M51] [M54] [M58] [M32] [M40]	20
	E9 Time-to-market	[M17] [M23] [M37] [M41] [M42] [M46] [M50] [M51] [M60]	9
	E10 When to implement decisions	[M8] [M17] [M3] [M21] [M24] [M22] [M25] [M29] [M42] [M47] [M53] [M32] [M9] [M61] [M40] [M15] [M1]	17
	E11 Technical debt evolution	[M6] [M45] [M30] [M17] [M3] [M23] [M22] [M31] [M33] [M36] [M46] [M47] [M55] [M26] [M61] [M44] [M15] [M39]	18
	E12 Technical debt visualization	[M45] [M8] [M30] [M4] [M5] [M31] [M52] [M54] [M57] [M28] [M26] [M61] [M14] [M44] [M40] [M15] [M16] [M39]	18

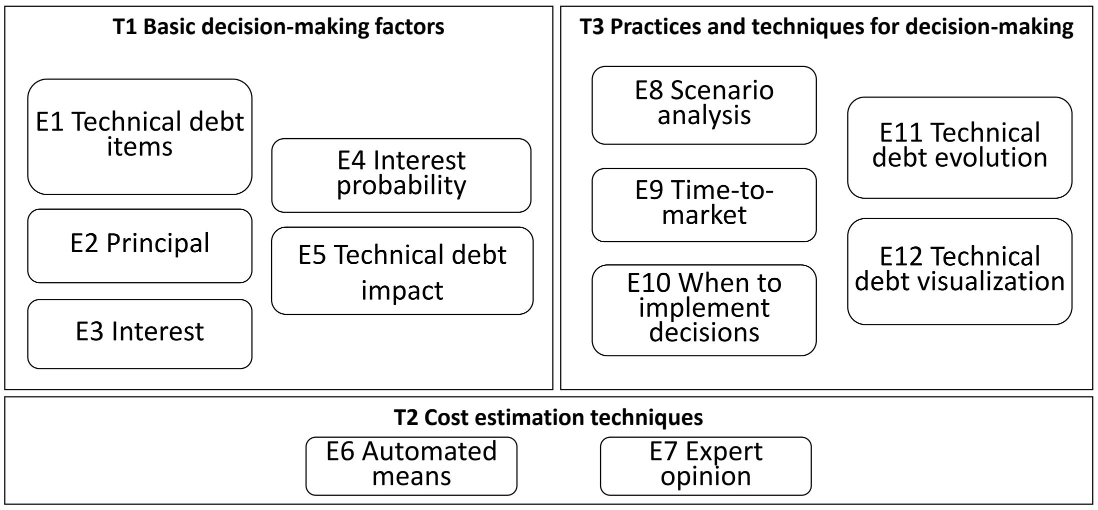

T1 Basic decision-making factors
E1 Technical debt items
E2 Principal
E3 Interest
E4 Interest probability
E5 Technical debt impact
T3 Practices and techniques for decision-making
E8 Scenario analysis
E9 Time-to-market
E10 When to implement decisions
E11 Technical debt evolution
E12 Technical debt visualization
T2 Cost estimation techniques
E6 Automated means
E7 Expert opinion

Figure 4.3: Taxonomy of the identified elements for technical debt management. Elements are grouped according to its use in technical debt management

uct development takes place. Therefore, in this chapter, the final list of stakeholders' points of view was as follows: (software) *P1 engineering*, *P2 engineering management*, and *P3 business organizational management*. *P1 engineering* includes concerns about processes such as software design and software construction, as well as, software architecture for some members of the community. *P2 engineering management* is focused on process planning and monitoring, including measurement. Finally, *P3 business organizational management* is focused on organizational goals, business strategy, time horizons, risk factors, financial constraints, and tax considerations. These points of view are aligned with the stakeholders identified by Yli-Huumo et al. [M61] in technical debt management: *development team*, *software architect* (*P1 engineering*); *team manager* (*P2 engineering management*); and *business stakeholder* (*P3 business organizational management*).

To analyze the elements of technical debt management from the stakeholders' points of view mapping was conducted. Figure 4.4 shows the mapping of the identified elements of technical debt management (see Section 4.4) and the stakeholders' points of view, which are described above.

The analysis revealed (see Figure 4.4) that most papers were focused on the *T1 basic decision-making factors*, that is, on obtaining the necessary information about the system's technical debt. In this case, the preponderant points of view were *P1 engineering* (e.g., the creation of source code, designing, architecting, or testing) and *P2 engineering management* (e.g., planning, or product quality management).

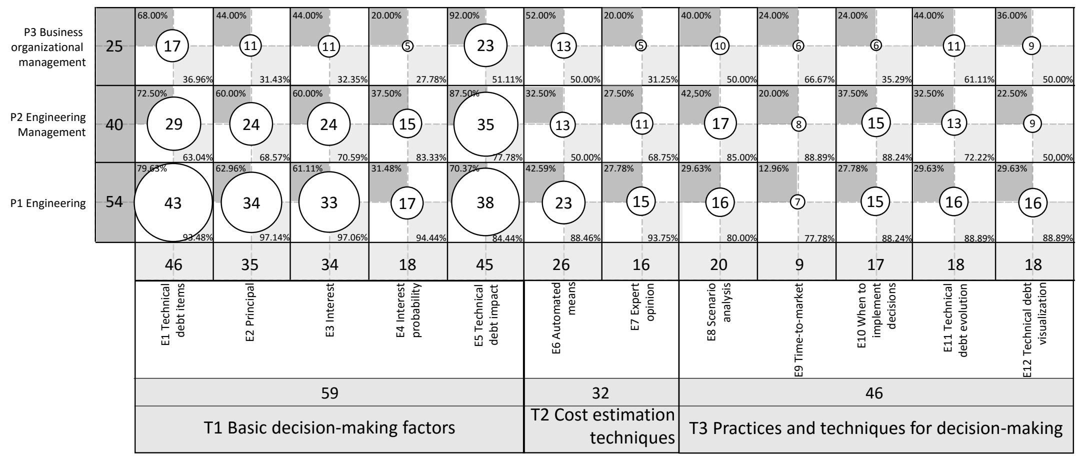


Figure 4.4: Mapping of the elements in the stakeholders' points of view. All the papers identified in this study are considered in the mapping. Mapping of the elements to support decision making in managing technical debt versus the engineering, engineering management, and business organizational management points of view. Each of the selected papers can include several elements and can be mapped onto more than one point of view. In each cell, the number in the center is the number of papers that identify an element from a specific stakeholders' point of view. The upper left percentage is the percentage of papers that identify the element (column) as specific to a point of view of (row), and the lower right percentage is the percentage of papers with the specific point of view (row) that identify the element (column). The first column shows the summary of the papers per stakeholder point of view, while in the bottom of the figure, there are summaries of papers per element and per type of element.

The papers included in *P1 engineering* and *P2 engineering management* strongly highlighted all the identified technical debt management elements. Conversely, this was not the case for *P3 business organizational management*, apart from *E9 time-to-market*, *E11 Technical debt evolution*, and *E6 automated means* elements. This finding indicated a business requirement for technical debt management, which could include the following: to quantify the impact of technical debt; to estimate technical debt automatically to avoid extra effort in the developments process; to know the effects of technical debt in the delivery capacity of the team putting at risk the ability to meet the projects' deadlines; and to control the evolution of technical debt over time. This finding, however, should be assessed in future studies.

Concerning the *E9 time-to-market* element, an interesting finding was that it is hardly ever taken into account by the current technical debt management tools and methods as discussed in [36]. This is an interesting paradox and a serious issue: whereas *E9 time-to-market* is one of the least referenced, used, and suggested *T3 practices and techniques in decision-making*, it is probably the most referenced cause of technical debt and according to [114] one of its most relevant antecedents. Therefore, managing technical debt without considering *E9 time-to-market* could lead to wrong decisions that could affect important deadlines in a project.

Most of the studies analyzed focused on *T1 basic decision-making factors.* This finding may indicate that technical debt management is still in an initial phase. It also means that data estimation techniques and metrics for the technical information normally extracted from source code, repositories, or tracking systems have not yet been developed, or, at least, they have not yet been made available to the software engineering community. Whereas *T1 basic decision-making factors* seem to be conceptually close to the project *P1 engineering* activities, *T3 practices and techniques for decision-making* are not far from the decision-making process and are therefore closer to *P2 engineering management* and *P3 business organizational management* than to *P1 engineering.* Whereas *P1 engineering* is focused on finding project's technical problems, *P2 engineering management* and *P3 business organizational management* extract information from sources such as strategic decisions about the architecture, product release dates, new contract signatures, budgets, or technologies provided by partners, in the areas of management and strategy. It is worthwhile to highlight that *E5 Technical debt impact* was the only highly referred element, regardless of the point of view that was considered. Therefore, *E5 Technical debt impact* was found to link the three points of view in addressing technical debt management concerns. This finding suggests that quantifying the effects of technical debt could form an excellent *communication* *channel* among the different stakeholders in a project. Moreover, this finding supports the previous conclusion about the importance of the *E5 Technical debt impact* element (Section 4.4.1.5).

Finally, the elements of *E6 automated means* and *E7 expert opinion* fit the type *T2 cost estimation techniques*. These two elements are complementary: *E6 automated means* element refers to extracting the required data without disturbing the normal activity of developers, whereas *E7 expert opinion* refers to using experts to provide the information that cannot be automatically extracted and to make decisions based on estimations.

4.6 Retrospective and Discussion

4.6.1 Identification and Definition of the Elements

This review study performed a systematic mapping of the current literature on technical debt management. Sixty-three papers were analyzed and 12 elements of managing technical debt were identified. The elements were classified into three types: *T1 basic decision-making factors*; *T2 cost estimation techniques*; and *T3 practices and techniques for decision-making*. This classification allowed us to use a top-down hierarchical approach. Previous contributions to the literature addressed topics that were considered necessary to manage technical debt, but it was not clear how these contributions could fit an overall view of the research. The present review study provided a taxonomy of the elements.

The elements and types of elements, even when they were related, differed from the activities identified by Li et al. [69]. Although the activities mentioned in [69] represented the steps that have to be performed to manage technical debt, elements are used during activities as inputs, outputs, or mechanisms.

Alves et al. [6] identified several management strategies. These strategies referred to concrete methods or techniques used to manage technical debt. These methods included some the sources in which the elements of the present review were identified. Therefore, the goals of the studies are different. Alves et al. identified the current methods for technical debt management, whereas the present review identified what these methods took into account in order to manage technical debt.

These relationships are interesting for studying how to integrate technical debt management into a software product roadmap similar to that described in [109]. Part of this roadmap is the software development process or the strategic planning. Therefore, a more detailed study of such relationships should be addressed in a future work.

4.6.2 Stakeholders' Points of View with Regard to the Elements

The identified elements were mapped to three different stakeholders' points of view (see Figure 4.4). These points of view comprise the activities involved in the software product development enterprise: *P1 engineering*, *P2 engineering management*, and *P3 business organizational management*.

This mapping allowed us to determine how different stakeholders considered the same elements. Other stakeholders sometimes considered different elements but in all cases from a different point of view.

The first finding showed that the business organizational perspective was neglected in the literature, which was a serious obstacle from the point of view of enterprise management. More papers were focused on *P1 engineering* and *P2 engineering management* than on *P3 business organizational management*. Companies make products either to sell to customers or to consume internally. In all cases, products must make sense from a business point of view as well as from a technical point of view.

The second finding was related to an important issue: communication among stakeholders. As described in Section 4.5, the element of *E5 Technical debt impact* was the only element that was highly referred to, regardless of the point of view considered. This finding suggests that quantifying the effects of estimating technical debt could be an excellent *communication channel* among different stakeholders. This opens the issue of how estimating the effects of technical debt could be represented so that stakeholders with different technical backgrounds could understand estimation of the effects and could discuss them effectively with each other.

Another remarkable finding was that *E9 time-to-market* was hardly ever taken into account in the methods used to manage technical debt. However, from the point of view of *P3 business organizational management*, only *E9 time-to-market*, *E11 Technical debt evolution*, and *E6 automated means* elements were highly considered. This finding is consistent with the results reported in previous studies [6] [36], which present an interesting paradox: whereas *E9 time-to-market* was one of the least referenced elements of technical debt management, it was one of the most relevant antecedents of technical debt [114]. Therefore, the lack of support for time-to-market in the current technical debt management literature could lead to wrong decisions.

4.6.3 Baseline for a Framework

The analysis described in sections 4.4 and 4.5 provides a baseline for defining what could be considered a framework for technical debt management. This framework would consist of a bi-dimensional schema at a high level of granularity. The dimensions would be groups of elements and stakeholders' points of view. At a second level of granularity, the schema could be considered to have six dimensions that corresponded to the three groups of elements (basic decision-making factors, cost estimation techniques, and the practices and techniques used in decision making) and the three Stakeholders' points of view (engineering, engineering management, and business organizational management). This framework would represent technical debt management as an integral job of the enterprise.

The application of this framework could be twofold: first, it would be possible to define purpose-oriented models for technical debt management (which elements are required for managing technical debt according to different objectives), when dimensions and groups were decided according to specific goals. A second application would be to determine how specific methods (or models) that were built outside framework guidelines could be applied according to their characteristics when they were analyzed according to the framework.

The use of this framework could show that some elements have special relevance whereas not always support of methods can be found in the literature. This is the case of *E9 time-to-market* (see Section 4.4.1.9 and Section 4.5). Scenario analysis is important because technical debt management depends on the context (see Section 4.4.1.2 and Section 4.4.1.3).

4.6.4 Technical Debt Management Decision Making

In the present review study, the main finding concerning technical debt management decision making was that it is context dependent (see Section 4.4.1.2 and Section 4.4.1.3, and the considerations below in the current section). The consequence of such context dependence is that without a clear solid definition of context and precise estimations of technical debt, estimations of effects are of little use.

From the point of view of decision making in technical debt management, the findings showed that the elements of type *T3 Practices and techniques for decision-making* were the most relevant. These elements are required to make informed decisions. It is necessary to identify and analyze different possible decisions (*E8 Scenario analysis*) in order to realize the possible consequences of such decisions. Therefore, in making decisions about technical debt it is not enough to have an overall picture of the system's technical debt. Without data about the evolution and trend in the amount of technical debt (*E11 Technical debt evolution*), decisions will be made without enough information. By analyzing the trend in the amount of technical debt, it would be possible to estimate when to invest in removing technical debt before the debt becomes too high to be managed. Both *E8 Scenario analysis* and *E11 Technical debt evolution* imply that a future time-frame is required to perform the analysis.

In practice, the complete picture is more complex because time is also a constraint. This was particularly highlighted by *E9 Time-to-market*, *E10 When to implement decisions*, and *E11 Technical debt evolution*. These elements imply that based on the constraint of time, more or less effort should be made to remove technical debt. For example, a scenario in which a startup is in a race to be the first to release a product in a market is different from a scenario in which a consolidated company has a product that leads the market, and therefore, it is possible to delay a new release to remove technical debt. Therefore, any decision made in technical debt management implies a trade-off between software release characteristics and technical debt removal.

Finally, even with effective tools and practices that identify, measure, and estimate the effects of technical debt, without the means to make such debt visible (*E12 Technical debt visualization*) it would be difficult for companies to understand the real situation of their software products. Visualization techniques are a means of providing fast and transparent communication. If decision-makers do not obtain technical debt information in a format that they understand, they could make wrong decisions.

4.6.5 Implications for Research

The results of the present study have several implications for the research on the management of technical debt, including the following:

- Further research is needed to integrate automatic data extraction with expert knowledge for technical debt management.
- Advancements are required to make trade-offs between new product characteristics and technical debt removal in new releases.
- Because a time frame is required to perform some analyses (*E8 Scenario analysis* and *E11 Technical debt evolution*, it is necessary to determine the appropriate time frame that considers all the related issues.
- There is a lack of research on the business perspective of technical debt. Further studies in this direction would make valuable contributions to the literature.
- There is a need to determine how to define contexts, and to use them in estimations.

- Visualization techniques are required to estimate technical debt and its effects. These techniques should be designed and built specially to take into account the different backgrounds of stakeholder.

- Research is required on the integration of technical debt management into the roadmap of the software product.

4.6.6 Implication for Practitioners

The results of the present study have several implications for practitioners, such as the following:

- A guide to study specific cases of technical debt management in relation to the identified elements should be produced. This issue pertains to technology transfer or standardization rather than research.

- Using the defined framework, organizations could create models of technical debt management by using the elements identified in this study.

- Practitioners could identify factors that have to be taken into account in making decisions about technical debt management.

- Organizations could compare their current practices in technical debt management with the elements of the described framework to identify gaps in their technical debt management process.

- Practitioners should consider that technical debt management is relevant to their work whenever they are working on a software product roadmap.

4.7 Threats to Validity

This section addresses potential biases and the actions taken to minimize their effects. To analyze potential biases in a more systematic way, in this section all the potential biases in systematic reviews were analyzed following the definitions given in [115]. According to these definitions, there are three main groups of biases: bias in identifying articles, bias in choosing studies, and bias in obtaining accurate data. In the following subsections, the biases in each group are analyzed.

4.7.1 Bias in identifying articles

Several factors can affect the identification of articles: the criteria of the reviewers and editors of journals or conferences, industry-sponsored research in some areas, place of publication, biased indexing studies in literature databases, inadequate or incomplete searches, articles that are cited more often than others are, and studies that generate multiple publications. In the present review, several different literature databases were used to include the maximum number of sources and to minimize the impact of the above-mentioned biases. Because of the complications involved in identifying that several publications are in fact results of the same study, no action was taken to merge publications. This is a minor risk because few papers included in this study were authored by the same researcher.

4.7.2 Choosing study biases

The process used to select the papers was conducted following the steps provided by Petersen et al. [84]. The inclusion and exclusion criteria could also be influenced by the personal biases of the author. The *a priori* definition of the criteria helped to minimize this potential bias.

4.7.3 Obtaining accurate data bias

In a literature review, the poor quality of sources can lead to inaccurate conclusions. To mitigate this potential bias, the selected papers were published in journals, conference proceedings, or workshop proceedings according to a peer-review process. To include all relevant studies about technical debt management, some book chapters were included. Because these were few in number compared to the other selected papers, the effect on the quality of the results was low. The author had to interpret the papers in order to classify them as being about engineering, engineering management, or business organizational management. This classification could be influenced by personal bias or by the information given in the papers, and this classification should be confirmed in subsequent research, such as by conducting interviews with practitioners.

4.8 Conclusions and Future Work

This chapter reports the findings, and conclusions of an analysis of the current literature on technical debt management. The focus of the chapter was to identify the elements of technical debt management. The research method used systematic mapping [84] and some synthesis activities. This chapter analyzed how current approaches supported the identified elements.

Based on the mapping conducted in this review study, one conclusion is that it was possible to define a framework. This framework could be used to produce specific decision-making models and methods or to assess existing ones (see Section 4.6.3). In contrast to the majority of the current approaches to technical debt management, the framework was not constrained by a concrete type of technical debt.

Another important conclusion is that technical debt is *context dependent* (see Section 4.6.4). This means that the context, which is difficult to define, must be part of the estimation model, includes issues such as the history of the product development, prospects, or time to market.

Introducing the business organizational perspective allowed to identify that most of the previous studies focused on the elements with engineering and engineering management points of view whereas the business organizational perspective was neglected. Within an industrial or government environment, this is a serious issue. It is important to highlight that while *E9 time-to-market* was one of the least suggested elements, it was probably the most referenced cause of technical debt. According to [114], it was one of its most relevant antecedents. This situation could lead to wrong decisions that could affect important deadlines in a project.

There were some indications that *E5 Technical debt impact* could be effective in allowing communication between the different stakeholders in a project (see Section 4.6.2). This would require both quantifying and visualizing the effects of technical debt.

The elements identified in the present review could be used to define models for technical debt management for specific systems with two objectives: to demonstrate how the elements work in practice; to implement specific technical debt management models based on the integration of the tools that are currently available for the management of technical debt.

In addition to the framework, an important finding of this chapter is, as is explained in Section 4.6.4, any decision about software evolution implies a trade-off between software release characteristics and technical debt removal.

Finally, to use the framework is necessary to know what is the support of tools for each framework's element. This is addressed in the following chapter.

4.9 Selected Publications

See Annex C

Part IV

Identification of tools and strategies that support the elements identified in Contribution 2 and the lacks in this support

Chapter 5

Tools and Strategies for Technical Debt Management

The goal of this chapter is to identify and analyze the tools and strategies for technical debt management that are available to support the elements identified in Chapter 4. To do that the systematic mapping study presented in Chapter 4 has been extended to analyze the tools and strategies proposed in the literature. This chapter is an excerpt from the following articles:

- C. Fernández-Sánchez, J. Garbajosa, C. Vidal and A. Yagüe, An Analysis of Techniques and Methods for Technical Debt Management: A Reflection from the Architecture Perspective, 2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics, in conjunction with the 37th International Conference on Software Engineering, Florence, 2015, pp. 22-28. [36].

  Copyright ©2015 IEEE.
- Carlos Fernández-Sánchez, Juan Garbajosa, Agustín Yagüe, Jennifer Pérez, Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study, Journal of Systems and Software, Volume 124, February 2017, Pages 22-38, ISSN 0164-1212. [39].

  Copyright ©2016 Elsevier Inc.

5.1 Introduction

This chapter extends the systematic mapping described in Chapter 4 by analyzing the selected studies that present tools or strategies used in technical debt management. The goal of this chapter is to identify the support of tools and strategies for using the framework described in Section 4.6.3.

The structure of this chapter is as follows: Section 5.2 presents the methodology used in this chapter. Section 5.3 presents some previous related studies and the differences with the present one. Section 5.4 presents the support of tools and strategies for using the identified elements described in Chapter 4. Section 5.5 shows the analysis of the rigor and relevance for the industry of the found tools and techniques. Section 5.6 discusses the results. And finally, Section 5.7 summarizes the outcomes of the case study in the context of this thesis.

5.2 Methodology

As mentioned above, this chapter extends a previously described systematic mapping. The details of the steps performed to conduct that systematic mapping can be seen in Chapter 4.

As part of the extension of the systematic mapping, the scientific rigor and the industrial relevance of the analyzed paper were assessed. The goal of this step was to know how the identified elements were supported in the currently available tools and techniques used in technical debt management from an industrial perspective.

The quality assessment of the reviewed papers is not covered by the systematic mapping approach [84]. We used the method proposed by Ivarsson and Gorschek [56] which was previously used with systematic mapping studies in the software engineering domain [81]. The model provides a set of rubrics to measure rigor and relevance for industry. Rigor refers to the precision or exactness of the research method used and how the study is presented. The model in [56] defines three aspects used to measure rigor: context described, study design described, and validity discussed. Each aspect is scored by 0, 0.5, or 1. Consequently, the total rigor score of a paper will be between 0 and 3: 0 is the worst score, and 3 the best score. A detailed explanation of the criteria used to assign each score is provided in [56].

Relevance refers to the realism of the environment in which the results are obtained and the degree to which the research method facilitates the transference of results to practitioners. The model defines four aspects that are scored by 0 or 1. Therefore, the total relevance score of a paper is between 0 and 4: 0 is the worst score, and 4 is the best score. A detailed explanation of the criteria used to assign each score is provided in [56].

5.2.1 Research questions

RQ1: What elements have been considered in the tools and strategies proposed to manage technical debt?

RQ2: What is the current support in industrial environments for the elements required in decision making in technical debt management in the form of tools and strategies?

5.3 Background and Related Work

Li et al. [69] identified tools for technical debt management whereas Alves et al. [6] identified technical debt indicators and strategies for technical debt management (approaches, methods, and models). The goal of the present chapter is different. The objective here is to analyze how tools and strategies of technical debt management support (or not) the elements required for technical debt management identified and defined in Chapter 4. Additionally, the rigor and relevance for the industry of the studies where those tools and strategies are presented were analyzed.

5.4 Tools and Strategies

To respond the first research question, a new mapping of the elements and stakeholders' points of view was performed. In that case using the papers that introduced one or several elements of technical debt management, and proposed tools or strategies to manage technical debt. Papers that simply introduced elements of technical debt management were excluded. Figure 5.1 shows this new mapping. The comparison of Figure 4.4 and Figure 5.1, shows an obvious difference in the number of papers on each element and point of view. As long as the set of papers used in Figure 5.1 was a subset of the set of papers used in Figure 4.4 the obtained difference in the number of papers could be expected. Nonetheless, the percentage of elements introduced with respect to the three points of view were proportional to those obtained in the entire selection of papers (see subsection 4.5).

This finding allowed us to conclude that the identified elements were not simply suggestions made by the authors but a set of elements used by those authors to define

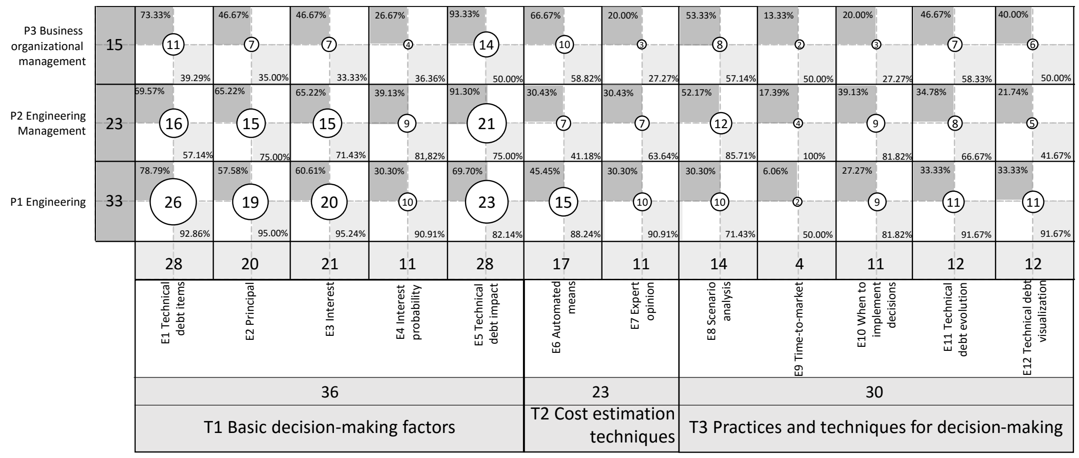


Figure 5.1: Mapping of the elements in the stakeholders' points of view. Exclusively the papers that defined methods for technical debt management were considered for this mapping. Mapping of elements to support decision making in managing technical debt versus the engineering, engineering management, and business organizational management points of view. Each selected paper can include several elements and can be mapped onto more than one point of view. In each cell, the number in the center is the number of papers that identify an element from a specific stakeholder's point of view. The upper left percentage is the percentage of papers that identify the element (column) as specific to a point of view of (row), and the lower right percentage is the percentage of papers with the specific point of view (row) that identify the element (column). The first column shows the summary of the papers per stakeholder point of view, while in the bottom of the figure, there are summaries of papers per element and per type of element.

concrete techniques for technical debt management. This second mapping was also useful to identify the coverage of the elements. It, therefore, was possible to determine some gaps in the current methods regarding elements that were not addressed to manage technical debt. The main shortcoming was the insufficient support shown for *E4 Interest probability*, *E7 Expert opinion*, *E9 time-to-market*, *E10 When to implement decisions*, *E11 Technical debt evolution*, and *E12 Technical debt visualization*. Based on this finding, it can be concluded that additional methods or improvements to the currently available methods are required to support these elements. Especially to take into account the business organizational management point of view.

5.4.1 Analysis of Tools and Strategies from the Elements for Technical Debt Management Perspective

5.4.1.1 E1 Technical debt items

Approaches to identifying items of technical debt are mainly focused on code debt and architectural debt. Approximately 65% of all code anomalies were related to 78% of all architecture problems [72]; therefore it could be thought that these techniques overlap. However, Zazworka et al. [M63] show that different techniques (modularity violations, code smells, grime, and automatic static analysis [ASA] issues) do not overlap. That is, they point different technical debt items with regard to maintainability including code and architectural debt.

Focusing on code technical debt, Nugroho et al. [M43] propose a method based on lines of code, code duplication, McCabe's cyclomatic complexity, parameter counts, and dependency counts to score software on the basis of its maintainability. This approach calculates the total technical debt in a system but does not identify concrete technical debt items. Other methods, use the combination of code metrics and thresholds to define detection strategies [M36], rules [M12], or quality requirements [M30].

One of the techniques analyzed by Zazworka et al. [M62] is focused on detecting code smells (e.g., God and brain code smells). As in the study of Zazworka et al., other studies show that a correlation exists between classes with code smells (or some of them) and change-prone, change size, change frequency, or error-prone classes [68, 62, 79, 80, 99, 125].

To identify architectural technical debt, Cai et al. [M8] use modularity violation detection (design rule violation) and rare class analysis to detect architectural debt items. For modularity violation detection, they use a tool called Clio [118] that allows one to detect files that change together when they are not supposed to be coupled. This tool uses a clustering technique to identify the system's modules, thus allowing one to analyze the modules' dependencies [119]. For rare class analysis, they use an algorithm to classify the files, considering their participation in patches created to fix detected bugs. A more recent tool, based on the same concepts of Clio, is Titan [122] which is based on the concept of design rule spaces [121]. Titan was used by Kazman et al for technical debt management [M28].

5.4.1.2 E2 Principal

Studies reveal two main strategies to estimate the principal. The first one is based on having a repository of similar changes and projects. Based on this accumulated knowledge, it is assumed that a similar problem in a similar project will imply the same effort to solve the problem. Following these criteria, in [M43] and [M13] a function of the estimated percentage of lines of code to be changed and an estimation of effort per line of code are used. Both variables are estimated using statistical information collected from other projects using the same technology. Curtis et al. [M12] detect code and architectural violations and use information obtained from several projects to estimate the effort needed to solve these kinds of violations, considering the programming language. An important consideration is that technical debt is *context* dependent [M54]. Therefore these approaches, based on statistical information collected from other projects, might introduce imprecision in the principal estimation.

The second strategy consists of detecting technical debt items and utilizing the typical effort estimation that the organization uses [M23, M20, M8, M62].

5.4.1.3 E3 Interest

For the interest estimation, some studies use an estimated maintenance effort based on information collected from other projects using the same technology [M43][M13]. Similar to the principal, an important consideration is that technical debt is *context dependent* [M54]. Therefore, due to these approaches use information collected from other projects, they might add imprecision in the interest estimation.

Others use defect likelihood and change likelihood to estimate the technical debt items' impact on system quality, for example, focusing on classes with the God class code smell [M62]. They calculate the defect likelihood on the basis of the times a technical debt item is changed to solve defects. Similarly, they calculate change likelihood on the basis of the number of changes performed in the technical debt item over time. In [M63], the analysis is extended to 30 indicators including modularity violations, several code smells and size to detect and analyze whose correlation with maintainability (defect likelihood and change likelihood).

Specifically focused on architectural debt, Cai et al. [M8] use variations in the cost-per-change and cost-per-defect to estimate the interest. They propose three proxy measures of effort for estimating the cost: actions, the number of commits/patches; churn, the number of lines changed in a file; and discussions, the number of textual comments about a file in the developers' discussions. They demonstrate that these three proxy measures of effort are valid, analyzing their correlation with other metrics that other authors have previously studied [71, 22]. However, they do not provide a concrete way of transforming the proxies' measure variations into maintenance effort variations.

Another proxy used to estimate interest is to monitor developers activity [M56]. Collecting metrics about the activities with the development environment that developers perform when they are working with classes shows the difference in maintenance efforts.

5.4.1.4 E4 Interest probability

Several studies propose assigning a probability to the interest estimation. In this way, the interest can be estimated as expected interest. Cai et al. [M8] use triangular distribution for the interest estimation that is, a pessimistic value, an optimistic value, and a most-likely value for the interest estimation. Several studies use the probability of change scenarios to model the interest probability [M33, M51, M20]. However, they do not provide concrete methods to estimate such values. In these methods, this estimation is delegated to project managers or architects. Therefore, it is a challenge how to estimate the probability distribution of interest to manage architectural technical debt.

5.4.1.5 E5 Technical debt impact

This section incorporates techniques that focus on the economic consequences of technical debt, perform some cost-benefit analysis on the basis of principal and interest, and/or provide ways for ranking technical debt items considering their impact on the system.

One strategy used to estimate technical debt's economic consequences is oriented to provide a big picture of the whole system without providing low-level detail of how technical debt is distributed in the system. Nugroho et al. [M43] propose a method based on code metrics (lines of code, code duplication, McCabe's cyclomatic complexity, parameter counts, and dependency counts) to score software on the basis of its maintainability. This same approach can be seen in [M13]. Based on the accumulated data of more than 170 systems, they provide estimations of the cost of change in order to increase the software's score in the ranking. Nevertheless, it only estimates the economic consequences at the whole-system level, while it does not identify the modules or components in which technical debt has accumulated and consequently, the technical debt distribution over the system.

Curtis et al. [M12] use average effort—considering the programming language—per type of code or architectural violation detected and the cost per hour to estimate the principal cost. This provides a general view of the system's technical debt, but they do not consider the interest or other aspects in their estimations. Similar to this solution, Letouzey and Ilkiewicz [M30] assign a remediation cost and a non-remediation index per each type of quality requirement (i.e., type of technical debt item) defined in their technical debt management model.

For architectural debt, Cai et al. [M8] use three file metrics to measure maintenance efforts. They use churn (lines of code changed in each change), actions (commits in which the file has been involved), and discussions (a metric based on the text mining of different sources, such as discussion forums and commit descriptions). They demonstrate that these three metrics are correlated with maintenance effort and proposed them to estimate the benefit of refactoring. However, a clear way of transforming the three metrics' values into a refactoring benefit is not shown. They also use the cost to solve defects and the defect rate as variables. Finally, they use real options to combine the different variables, performing an economic analysis of the future costs and benefits of the system. Similarly, Alzaghoul and Bahsoon [M2] analyze Web service selection using real options. They consider the technical debt generated when the selected web services do not have enough scalability to support the system's future growth, but they also consider the importance of not wasting resources due to excess of scalability capabilities.

Several authors use cost-benefit analysis to estimate technical debt's impact. The basic way to perform a cost-benefit analysis is to compare the cost of removing technical debt (principal) with the benefit obtained (normally, the avoided cost due to not having interest) [M24, M30, M28].

Some of these methods are oriented to release planning [M42]. Their goal is to identify when it is more profitable to initially invest in architecture or when it is better to release functional features as soon as possible. That is, this method is oriented to decide when it is better to incur technical debt by adding features as soon as possible or when it is better to add features later.

Some authors add time as a variable to be considered in the analysis. Therefore, they take into account technical debt's possible evolutions over time [M23, M43, M13]. Other authors use real options to perform the analysis [M8, M40], in this case using a valuation technique based on Monte Carlo simulations. Also, considering the time frame, other authors use decision trees to perform the cost-benefit analysis [M20].

Authors use the estimated cost-benefit ratio to perform a ranking [M62][M30]. This ranking is useful when it is necessary to solve the technical debt items with the highest priority. The ranking is dependent on the cost-benefit analysis method. Therefore, only the variables considered in such a method are used to perform the ranking. Another method that Guo and Seaman use [M24] consists of applying a model based on the portfolio approach. Portfolio management comes from the finance domain and focuses on selecting the assets that maximize return on investment or minimize investment risk [M24].

Several challenges persist. In order to have a realistic cost-benefit analysis, it is necessary to evaluate more than just principal and interest. Many times, authors point out that technical debt is originated when time-to-market restrictions are present. Therefore, the costs of delaying a functionality or release in order to remove technical debt have to be considered. Additionally, the team's capacity to perform tasks should be considered because it is limited.

5.4.1.6 E6 Automated means

Many authors use tools for automatically detecting sources of technical debt (see Section 5.4.1.1). Also, historical data can be used to estimate the interest for a specific organization or project [M17]. Some authors define code and file metrics used to estimate interest that can be extracted automatically from source code (see Section 5.4.1.3). Usually, automated estimates include code/file metrics, effort measures, and the files' evolutionary history over the system's different versions [M8, M28].

Two approaches can be identified: one based on having a historical repository of projects with similar characteristics and a second based on mining a project's available resources (source code, control version systems, etc).

5.4.1.7 E7 Expert opinion

Almost all of the authors suggest the need to use expert knowledge to add information that cannot be estimated in another way. Also, Cai et al. [M8] propose a decision-support system for architectural refactoring decisions. Their goal is to give refactoring recommendations to experts (the project manager or the architect). Then, the expert could analyze different scenarios on the basis of the estimations to decide which recommendation to follow.

5.4.1.8 E8 Scenario analysis

The authors use different kinds of scenarios: (1) scenarios to analyze technical debt goals and estimate the effort required to achieve them [M30]; (2) release scenarios to analyze the most profitable release path based on the architectural technical debt incurred [M42][M25]; (3) what-if scenarios used to provide managers or architects with the possibility of seeing the estimated impact of different decisions regarding refactoring the architecture in order to improve the software's modularity [M8]; (4) and change scenarios to analyze the possible evolution of the technical debt in the system [M20].

None of the proposed methods analyze several types of scenarios. Furthermore, the methods that use scenarios as part of their analyses focus on concrete kinds of technical debt or use different detection strategies that make them difficult to integrate. Further effort is needed to define methods for estimating architectural debt that allow one to combine all of the types of scenarios.

5.4.1.9 E9 Time-to-market

Some studies consider time-to-market as a restriction when they define scenarios and as a potential benefit of incurring technical debt [M23, M42, M51]. However, they do not provide explicit methods for considering time-to-market in order to manage technical debt. Time-to-market is essential to the success of many projects and products. Therefore, it should be considered explicitly in making a decision about when to delay groups of features or when to remove technical debt.

5.4.1.10 E10 When to implement decisions

Several authors identify release planning as the step during which decisions about technical debt management can be executed [M24][M42][M25]. Two different decisions have been identified. The first consists of determining when it is necessary to reduce technical debt [M24]. The second is oriented to decide whether it is better to implement features as soon as possible (this implies future reworks due to adaptations) or expend some time in architectural tasks and then implement the features. [M42].

Real options can help with deciding when refactoring is profitable [M8]. If the case is an option-based approach some important estimations are usually required: data inferred from evolution history and the prediction of future changes and cost estimations [M8]. Other authors have suggested using portfolio theory with the same objective [M24].

5.4.1.11 E11 Technical debt evolution

Many articles use project's historical data to estimate the interest (see Section 5.4.1.3) based on the evolution of some code or file metrics. However, only a few consider technical debt's evolution over time. Some authors analyze technical debt' evolution over the project releases [M23, M39, M55]. Also, Marinescu [M36] defines an indicator for tracking technical debt's evolution over the project releases. These indicators

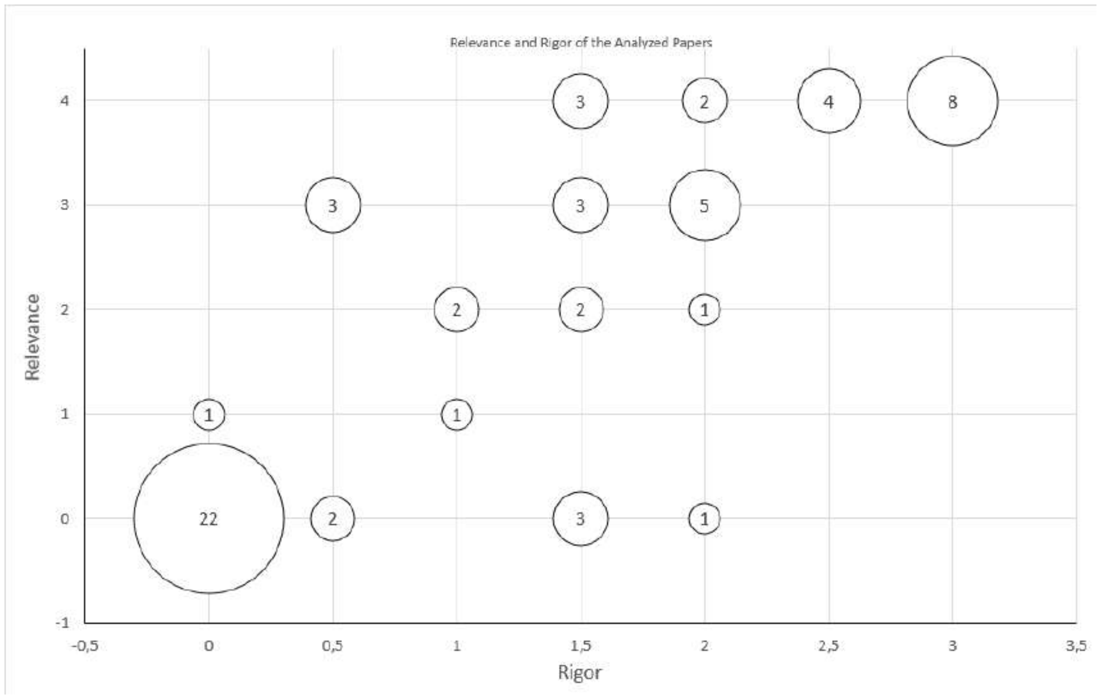


Figure 5.2: Mapping of selected papers with respect to relevance and rigor scores as defined in Section 5.2.

provide information about how technical debt is rising or not rising in the system, but they do not provide information about whether or not accumulating technical debt has additional consequences.

5.4.1.12 E12 Technical debt visualization

In general, few methods exist for visualizing technical debt. Most studies show charts featuring the relationship among principal, interest, and/or time. Other studies use design structure matrix (DSM) to show different kinds of relationships between the software modules [M5, M28], or dashboards in order to make visible the proportion of lines of code that exceed the quality requirement established [M30, M39]. Finally, some studies show the components more affected by technical debt [M4, M14]

5.5 Technical Debt Management in the Industrial Environment

The method described in Section 5.2 was applied to analyze the extent to which the current techniques used for technical debt management are relevant in an industrial environment. This method provides two scores to quantify the rigor and the relevance of the studies from an industrial perspective. The method used to score rigor and relevance (see Section 5.2) focused on measuring whether the techniques and methods have been validated. Therefore, papers that did not provide data about the validation of technical debt management methods obtained low scores.

Figure 5.2 shows the scores of the analyzed papers. Most papers did not provide enough details about the methods or techniques to be relevant to the industry. That is, they did not provide enough details to allow an independent company to use the method proposed. However, as shown in Figure 5.3, in recent years, the trend has been that the average rigor and relevance of the research on technical debt management has increased. Therefore, the technical debt research community is aware of this lack of rigor and relevance in the industry, and it is working to improve the situation.

Figure 5.4 shows the mapping of the elements and points of view in which papers that presented methods or techniques for technical debt management that had rigor and relevance greater than a threshold were considered. In the present analysis, the threshold for rigor is equal to 2, that is, only papers with a rigor score of 2 or more were considered. This value was chosen so that only papers with a score at least equal to the half-maximum possible score were included. Following the same criteria, the threshold for relevance was set at 3. Many of the elements counted on having only two or three methods with enough rigor and relevance for industry. The elements *E1 Technical debt items* and *E5 Technical debt impact* were outstanding because they had a consistent support that was independent of the point of view.

Therefore, based on these findings, it can be concluded that more tools and strategies, or more detailed reports of the current ones, are necessary to support their use by software companies in managing technical debt.

5.6 Findings

The elements are used in tools and strategies. Therefore, they are not simply suggestions, but considerations used when managing technical debt. However, there are not tools or strategies that support all the elements. Hence, it is necessary to study how

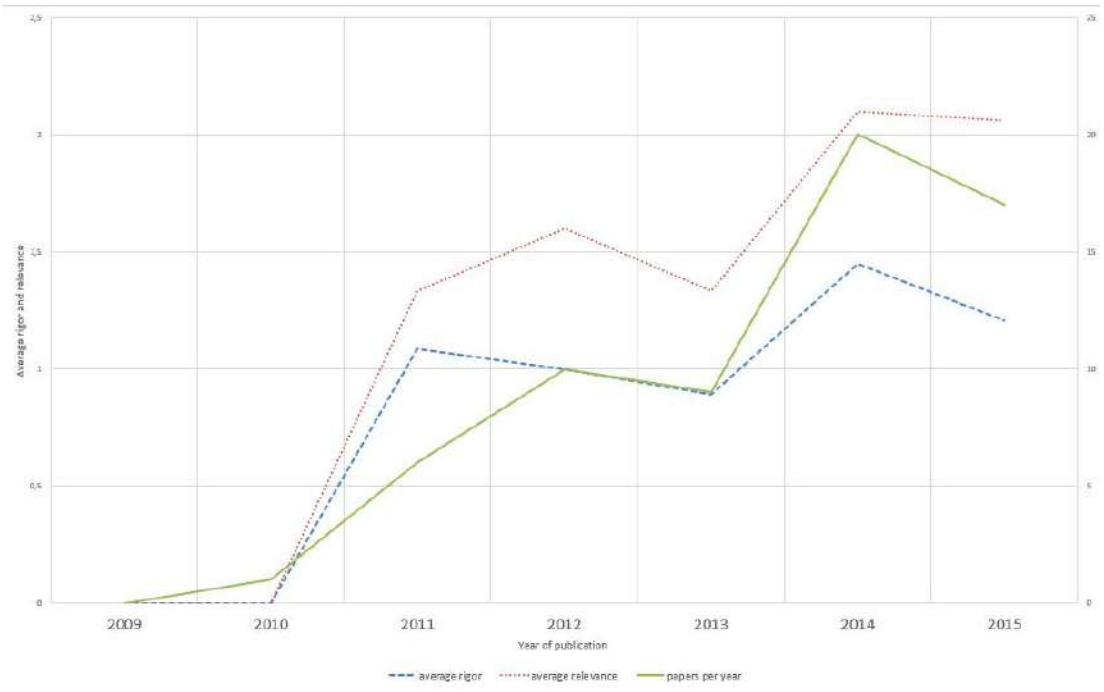
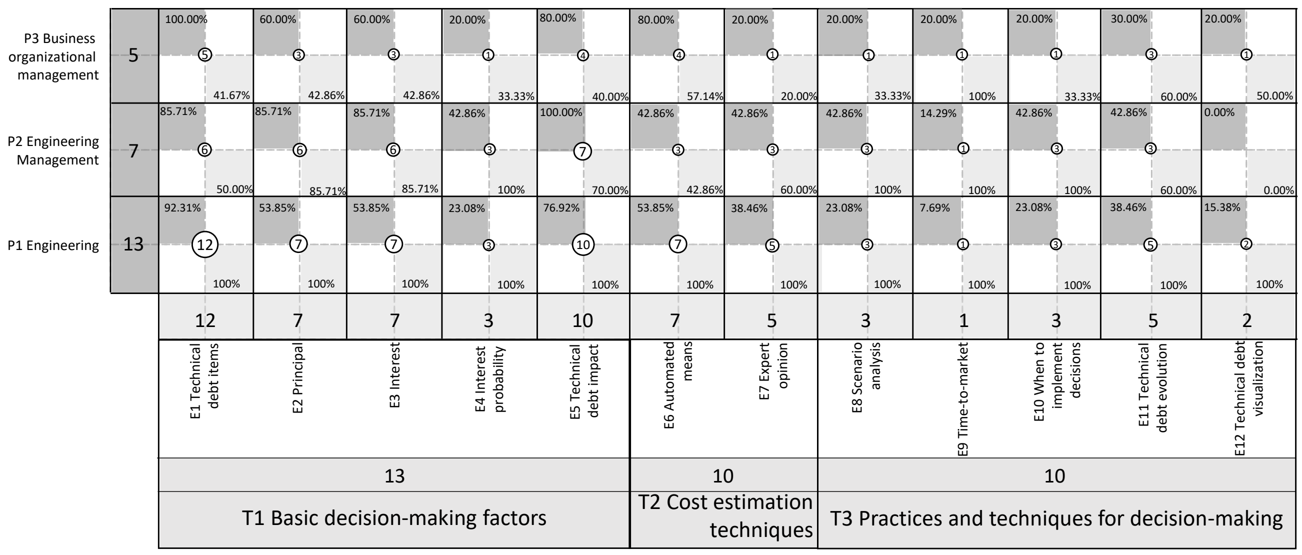


Figure 5.3: Temporal evolution of the number of papers with respect to the average relevance and the average rigor scores.



		E1 Technical debt items	E2 Principal	E3 Interest	E4 Interest probability	E5 Technical debt impact	E6 Automated means	E7 Expert opinion	E8 Scenario analysis	E9 Time-to-market	E10 When to implement decisions	E11 Technical debt evolution	E12 Technical debt visualization
P3 Business organizational management	5	100.00% 5 41.67%	60.00% 3 42.86%	60.00% 3 42.86%	20.00% 1 33.33%	80.00% 4 40.00%	80.00% 4 57.14%	20.00% 1 20.00%	20.00% 1 33.33%	20.00% 1 100%	20.00% 1 33.33%	30.00% 3 60.00%	20.00% 1 50.00%
P2 Engineering Management	7	85.71% 6 50.00%	85.71% 6 85.71%	85.71% 6 85.71%	42.86% 3 100%	100.00% 7 70.00%	42.86% 3 42.86%	42.86% 3 60.00%	42.86% 3 100%	14.29% 1 100%	42.86% 3 100%	42.86% 3 60.00%	0.00% 0.00%
P1 Engineering	13	92.31% 12 100%	53.85% 7 100%	53.85% 7 100%	23.08% 3 100%	76.92% 10 100%	53.85% 7 100%	38.46% 5 100%	23.08% 3 100%	7.69% 1 100%	23.08% 3 100%	38.46% 5 100%	15.38% 2 100%
		12	7	7	3	10	7	5	3	1	3	5	2
		13					10		10				
		T1 Basic decision-making factors					T2 Cost estimation techniques		T3 Practices and techniques for decision-making				

Figure 5.4: Mapping of the elements to stakeholders' points of view to assess industrial rigor and relevance. Only papers that scored rigor equal to or more than 2, and relevance equal to 3 or more were considered. Mapping of elements to support decision making in managing technical debt versus the engineering, engineering management, and business organizational management points of view. Each selected paper can include several elements and can be mapped onto more than one point of view. In each cell, the number in the center is the number of papers that identify an element from a specific stakeholder's point of view. The upper left percentage is the percentage of papers that identify the element (column) as specific to a point of view of (row), and the lower right percentage is the percentage of papers with the specific point of view (row) that identify the element (column). The first column shows the summary of the papers per stakeholder point of view, while in the bottom of the figure, there are summaries of papers per element and per type of element.

to integrate tools and strategies to work together.

To determine the relevance for the industry of the studies with regard to technical debt management each paper was analyzed using the method described in Section 5.2. The results showed that most papers did not introduce methods or techniques in enough detail to be relevant to the industry. Nevertheless, in recent years, the relevance and rigor of the papers about technical debt management have increased. By analyzing the mapping of papers with high rigor and relevance (see Figure 5.4), it was possible to determine that more methods, or more detailed reports of the current tools and strategies, are necessary to support software companies for using them to manage technical debt. Some elements required more support in the tools and strategies used for technical debt management: *E4 interest probability*, *E7 expert opinion*, *E9 time-to-market*, *E10 When to implement decisions*, *E11 Technical debt evolution*, and *E12 Technical debt visualization*. This finding was particularly apparent in *E12 Technical debt visualization*, which is consistent with the results in Alves et al. [6]. Finally, as can be seen in Figure 5.4, business organizational management is less supported than the engineering and engineering management points of view.

5.7 Conclusion

The available techniques for technical debt management identified in the current literature have been analyzed.

This analysis shows that further studies are necessary to fully support technical debt management and also essential elements not currently covered, such as *E9 time-to-market*. Other elements as *E4 interest probability*, *E7 expert opinion*, *E10 When to implement decisions*, *E11 Technical debt evolution*, and *E12 Technical debt visualization* also require more support, especially from the business organizational management point of view.

Different strategies are focused on different elements. There are not tools or strategies that support all the elements. Therefore, it is necessary to go further in the integration of tools and strategies to manage effectively technical debt.

However, there is not evidence of how the different tools and strategies can be integrated. Therefore, any effort in this direction has to consider the possibility of experimenting with different strategies and tools to analyze the benefit of using them.

Chapter 6

TEDMA Tool: A Tool for Technical Debt Management

This chapter presents TEDMA Tool, a tool for technical debt management. TEDMA implementation responds to the necessity of a tool that facilitates the experimentation with different technical debt management tools and strategies. Therefore, TEDMA facilitates the integration of third-party tools and the definition of models for technical debt management. This necessity was identified as a finding of the analysis of the tools and strategies for managing technical debt performed in Chapter 5. This chapter is an excerpt from the following paper:

Carlos Fernández-Sánchez, Juan Garbajosa, Héctor Humanes, Jessica Díaz, An Open Tool for Assisting in Technical Debt Management, submitted to Euromicro DSD/SEAA 2017.

6.1 Introduction

Chapter 5 analyzes the support of tools and strategies for technical debt management elements identified in Chapter 4. The present chapter presents TEDMA, a tool implemented with the goal of integrating existing tools to be able to use the framework based on the technical debt management elements in real projects. To do that, TEDMA has as the main goal to facilitate the integration of different tools and strategies to use their outcomes to manage technical debt. In that way, TEDMA provides a platform for empirical experimentation of techniques for technical debt management. TEDMA capabilities are important in this thesis to be able to perform the case study described in Chapter 7.

The remainder of the chapter is organized as follows: Section 6.2 will describe TEDMA. Section 6.3 will analyze TEDMA with other existing tools for technical debt management. Section 6.4 will discuss some of the goals achieved by TEDMA. And finally, Section 6.5 will presents the conclusions of the chapter.

6.2 TEDMA Tool Description

6.2.1 Overall View

To understand the architecture of the TEDMA tool it firstly may help to describe the process to analyze a project. Figure 6.1 depicts the life cycle of a project in TEDMA. A project has to be added to the tool by providing a name, a description, and the location of the source code repository (local or remote). This creates a project-core that allows us to work with the project. After that, the next step is to load the basic project data into the tool's database. This information is mainly source code data about changes in files over the evolution of the project. Once this basic information is obtained, the project can be analyzed by any of the available analyzers. An analyzer is an abstraction of any tool or technique used to obtain relevant data for technical debt management. When new changes are uploaded to the repository, those changes can be loaded into the tool and analyzed. At any moment a project can be removed from TEDMA for releasing the resources used in the databases.

Figure 6.2 depicts how the information is stored in TEDMA. TEDMA stores information for each revision and for each file in each revision. Finally, for each file, several metrics are stored. Examples of metrics are basic size metrics of each file as size in bytes, the number of lines, and, if the file has changed, the type and size of the change. Depending on the analyzers executed, different metrics can be stored. Each analyzer

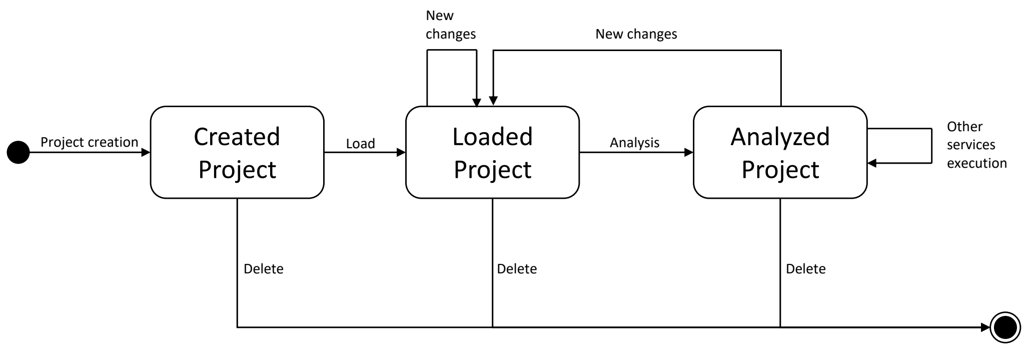


Figure 6.1: Life cycle of a project in TEDMA

prescribes how its output is stored. For example, the current implementation of the PMD analyzer [85], one of the tools integrated, adds a problem for each file in which PMD detects problems. Figure 6.3 shows examples of the information stored.

Additionally, TEDMA ensures that any change or action over the files is considered just one time. This is important in merging revisions where changes could be considered twice if they are not carefully checked.

6.2.2 Obtaining information from projects

Currently, TEDMA can gather information about the evolution of each file in a project, and this information is mainly obtained from git repositories [43], PMD [85] detected code smells, and Findbugs [40] detected problems. TEDMA can analyze and store information using PMD and Findbugs [40] to analyze each revision of the projects. Both, PMD and Findbugs analyzers are limited to analyze Java projects. In the case of Findbugs, each release has to be compiled, so it is necessary to have installed additional software to build the project, for example, Maven and Gradle. Other metrics have been directly implemented, for instance, the probability of change and expected size of change as are defined in [126]. Therefore, the current information managed by TEDMA is mainly based on Git, PMD detected code smells, and Findbugs detected problems. Data are collected for all the releases of the projects so that metrics evolution can be analyzed. Figure 6.4 shows an example of the data that is generated, in that case, using R [89] to generate the graphical representation using the data exported by TEDMA. R is a free software environment for statistical computing and graphics. R and its libraries implement a wide variety of statistical and graphical techniques. In the following sections, it is explained how R has been used for other purposes in TEDMA.

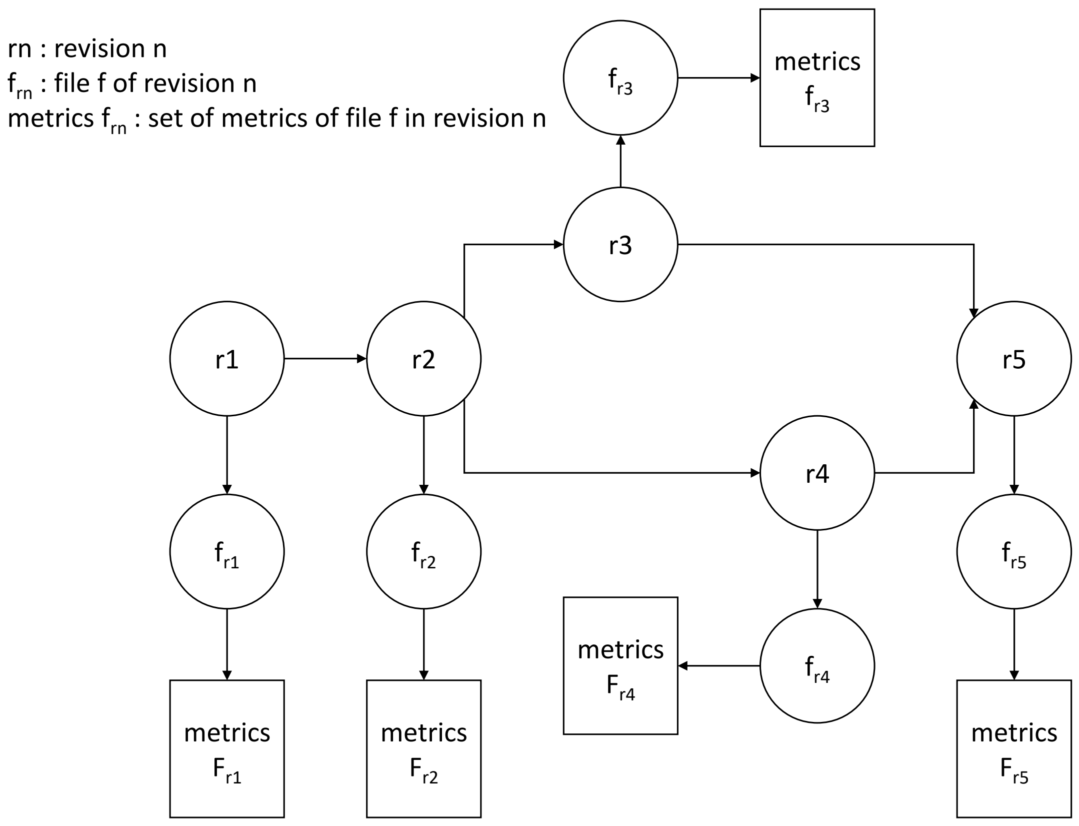
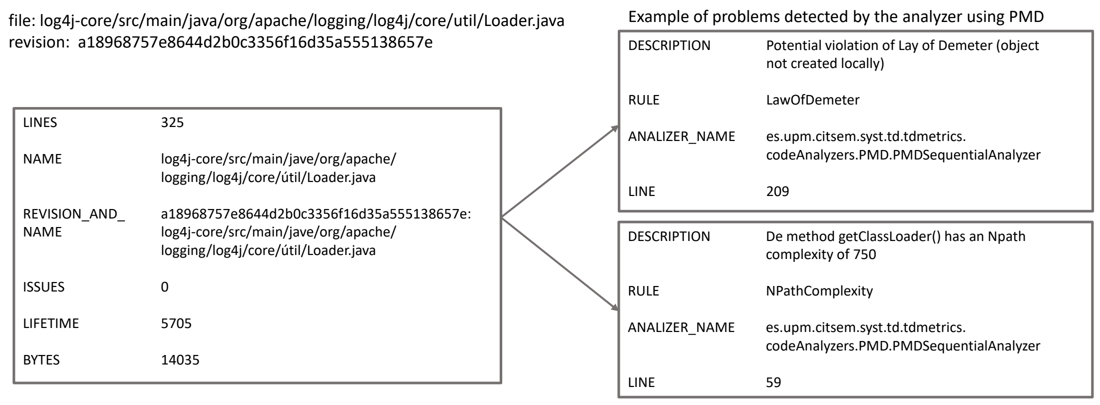


Figure 6.2: Basic data structure


file: log4j-core/src/main/java/org/apache/logging/log4j/core/util/Loader.java
revision: a18968757e8644d2b0c3356f16d35a555138657e

LINES	325
NAME	log4j-core/src/main/jave/org/apache/logging/log4j/core/útil/Loader.java
REVISION_AND_NAME	a18968757e8644d2b0c3356f16d35a555138657e: log4j-core/src/main/jave/org/apache/logging/log4j/core/útil/Loader.java
ISSUES	0
LIFETIME	5705
BYTES	14035

Example of problems detected by the analyzer using PMD

DESCRIPTION	Potential violation of Lay of Demeter (object not created locally)
RULE	LawOfDemeter
ANALIZER_NAME	es.upm.citsem.syst.td.tdmetrics.codeAnalyzers.PMD.PMDSequentialAnalyzer
LINE	209

DESCRIPTION	De method getClassLoader() has an Npath complexity of 750
RULE	NPathComplexity
ANALIZER_NAME	es.upm.citsem.syst.td.tdmetrics.codeAnalyzers.PMD.PMDSequentialAnalyzer
LINE	59

Figure 6.3: Example of metrics stored by TEDMA and problems detected by the analyzer that integrates PMD.

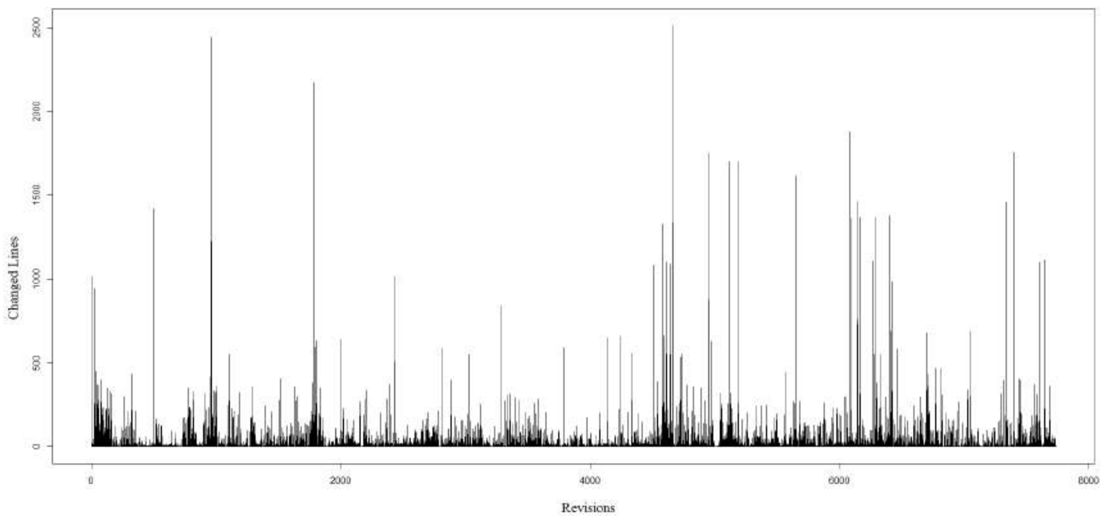


Figure 6.4: Changed lines in approximately 1500 files in Apache Log4j 2 project over near 8000 revisions

Figure 6.4 shows a graphical representation of several time series. Each time series represents the number of lines changed in a file. This kind of graphic representation is useful to see how the behavior of the project changes and to identify outliers.

Figure 6.5 shows the evolution of one metric, in this case, the number of files with at least a method with a cyclomatic complexity issue as it is detected by the default PMD configuration in Apache Log4j 2 project. This type of time series provides information about how some problems are evolving in the analyzed projects. In this case, it seems that the project is getting more complex over time. This type of analysis can help to identify potential technical debt problems over the evolution of projects.

6.2.3 Processing information from projects

Currently, the TEDMA core components are already implemented. TEDMA is being used to analyze open source projects on GitHub that are widely used in the software development industry. Examples of already analyzed projects are Spring-Framework, Karaf, Log4j 2, Hbase, and Hadoop. Thanks to these analyses and that projects are really big, it was possible to test the capabilities of TEDMA. In this sense, TEDMA has proved helpful to investigate metrics and models for technical debt management. Currently, it is been used with that goal. It provides empirical data obtained from real projects used in the software development industry. This kind of tools is necessary because it allows the experimentation with research findings in development environments

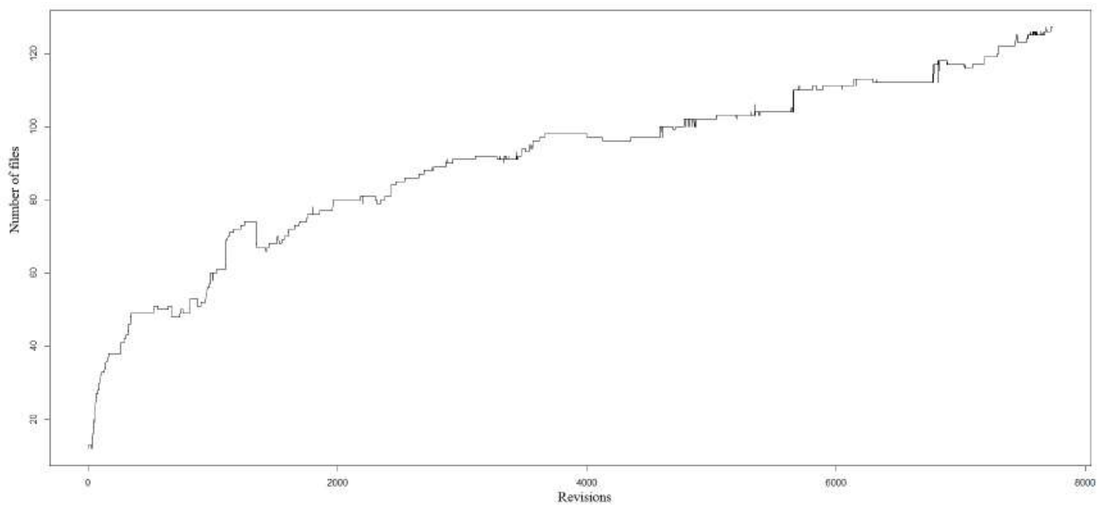


Figure 6.5: Evolution of the number of files with cyclomatic complexity issues in Apache Log4j 2 project

and assists the collection of evidences to support further research [102].

The tool provides access to data at different abstraction levels that can be combined when needed. The most abstract level is the entity level (project, revision, file, change, etc.). At that level of abstraction, the programmer can manage all these concepts without thinking in the source code repository or the database. If it is needed, the TEDMA API supports the direct usage of the Neo4j [76] API to manage the graph database and the JGit [58] API to manage the Git repository.

6.2.4 How TEDMA is built

In this section, we describe the main modules of TDManger and the role that they have in the tool. Figure 6.6 shows the main modules of TEDMA.

6.2.4.1 Data Layer

TEDMA uses several means to obtain and store projects data. The main data source is the source code repository. Currently, TEDMA only supports Git repositories. For each project, TEDMA clones the source code repository to be used in the whole life cycle of the project technical debt management.

After the source code repository, the most relevant data are stored is the graph database. For each project, TEDMA creates and maintains a graph database that represents the whole project evolution, including all the project files evolution. This

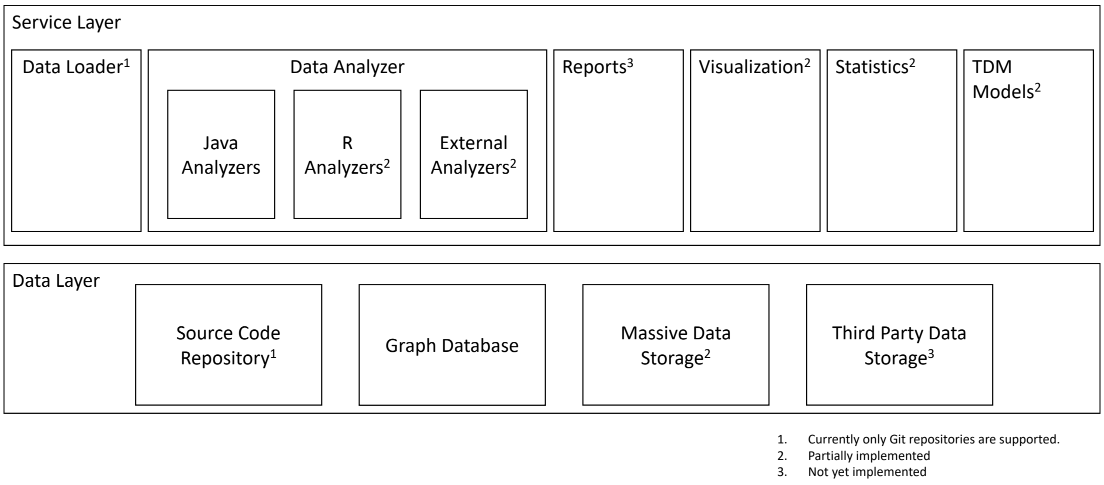


Figure 6.6: Modules of TDManger

graph allows accessing the information following the evolution paths of the project. Therefore it can be traversed using releases, revisions, and files following any combination of the stored relationships of such nodes. In fact, the graph database acts as an index for the rest of the information. If any node requires a large amount of data to be stored, an additional storage is used to avoid overcharging the graph with data that it is only required in specific analysis or reports. Additionally, it is planned to provide means to use third-party data storage by including extensions for other data sources, for example, external metrics databases.

6.2.4.2 Service Layer

TEDMA provides a set of services that can be used in the technical debt management process. The first service is Data Loader. This service is, in fact, the service that incorporates a project to the tool to be analyzed. This service is part of the core of TEDMA because it has the responsibility of cloning project repositories and creating graph databases with the basic project data that are required for the remaining services.

The data analyzer service allows to implement and execute different analyzers in the projects. As it was said before, an analyzer is an abstraction of any tool or technique used to obtain relevant data for technical debt management. The analyzers can be implemented using Java and R. Currently, two analyzers are available to use PMD [85] and Findbugs [40] to analyze projects. Other metrics have been directly implemented, for instance, the probability of change and expected size of change as are defined in [126]. All these implementations were done to demonstrate the integration capacity of TEDMA.

Reports service, which is not currently available, will be focused on the elaboration of automatic reports to show the results of the analyses. The reports will be oriented to different stakeholders roles.

Visualization service is in charge of showing dashboards and graphical representations of the results. It is related to the reports service but focused on interactive means of visualization of the information. This service is currently under development.

Statistics service is based on the integration of R in the core of the TEDMA. This facilitates the usage of R scripts and incorporates all the power of R to perform statistical analysis over the collected metrics. Currently, this service is under development. R can be used into the TEDMA using two different approaches, renjin [91] and Rserve [93]. Renjin is an interpreter of R that runs in the Java Virtual Machine. Therefore, due to TEDMA is implemented in Java the integration is easier and the communication between TEDMA and Renjin is more efficient. Unluckily, Renjin does not support some important libraries of R because they have to be implemented in Java. Rserve is a TCP/IP server which allows other programs to use facilities of R. In that case, the communication is less efficient than with Renjin but Rserve supports more R libraries than Renjin. However, Renjin is being continuously improved by its community. Hence, in the future, it is probably that Rserve will be not used by TEDMA.

Technical debt management models service corresponds with the goal of testing and developing technical debt management models that help make decisions about technical debt. Models can be defined in Java or R languages. Currently, a model for technical debt management is implemented in R, but not completely integrated into TEDMA. To be used it is necessary to export project data from TEDMA using an ad-hoc export module, and after that, it is possible to use the model from R directly. When the integration of R into TEDMA is complete, this will be done directly into TEDMA reducing the steps required.

6.2.5 Integration of third-party tools

To be extended, TEDMA provides one API to manage all the information stored in the Data Layer. The API allows working with git repositories using internally JGit [58]. The graph database is implemented using Neo4j [76]. The API provided by TEDMA uses internally Neo4j and it has different levels of abstractions that allow working with files, revisions, projects, etc., including the usage of queries using CYPHER language. This provides a big flexibility when new extensions have to be created. If it is needed, the TEDMA API allows using the Neo4j and JGit APIs directly.

This API can be used to extend any of the services provided by TEDMA or implement new ones. The API can also be used to export data or sets of data in the desired format to be used by external tools, for example, for statistical analysis or data mining purposes. Currently, to provide services it is necessary to implement some Java interfaces or to inherit from the abstract classes that implement the common functionality.

The level of external tools integration will depend on the characteristics of the tools to be integrated. For example, PMD and Firebugs were integrated using APIs that they provide.

6.2.6 TEDMA Tool Roadmap

The great advantage of TEDMA is that it can be integrated with other tools to obtain metrics from many different sources. At the moment TEDMA is used to analyze projects with thousands of revisions and thousands of files in each revision. This demonstrates that the core of TEDMA is ready for the analysis of real projects to extract metrics over their evolution.

The next planned integration is with SonarQube to take advantage of the great number of metrics that this tool can obtain. Additionally, SonarQube is widely used in software development industry, so this integration will help to use TEDMA in any development environment where SonarQube is currently been used.

Summing up, in the next months it is planned to extend TEDMA to implement all the modules described in Figure 6.6. The next specific features that will be included in TEDMA are:

- Support for SVN source code repository.
- Integration with SonarQube.
- Full integration of R.
- Visualization service.
- Report service.

When all these features are implemented TEDMA will be released. At that point, TEDMA will provide much more value for software development organizations by allowing them to use already implemented technical debt management models and by allowing them to implement their own models. The goal is to have a tool that can be used to generate reports and dashboards without a deeper knowledge of how TEDMA is working internally.

6.3 Related Work

To the knowledge of the authors, there is not another tool with the goal of integration of technical debt management tools. The available tools are focused on implementing metrics, specific techniques for technical debt management, or both of them. Techniques and tools for technical debt have been analyzed in literature reviews previously [37, 69, 39].

The goal of this tools is not to be a quality tool that implement metrics to keep the code quality. TEDMA pursues to manage technical debt by integrating other available tools that implements those metrics or technical debt strategies.

One of the most used tools to analyze code is SonarQube [107]. This tool integrates many code metrics and is widely used in software projects. It is based on the static analysis of source code. The goal of the tool presented in this paper is to analyze the evolution of code and to compare different techniques to analyze software. Therefore, one of our future work will include the integration with SonarQube to use it as another source of metrics for TEDMA.

Another tool, Titan [59], which provides mechanisms to estimate technical debt and that also provides a framework to make decisions based on that calculus, could be also integrated on TEDMA. Titan [59] is mainly focused on the analysis of modularity violations, similar to Clio [118], and a model for technical debt management. As the goal of TEDMA is to integrate both metrics and management models, the integration of a tool as Titan could be a good example of the capabilities of TEDMA.

6.4 Discussion

TEDMA is a tool to analyze software projects with a perspective of evolution. It is thought to work with projects with thousands of revisions and thousands of files per revision. To do that, it supports the integration of third-party tools for technical debt management in order to use them in the analysis of the evolution of technical debt of projects. It can be extended to include additional tools or to implement specific metrics. With these capabilities, TEDMA is a useful tool for researching using empirical data extracted from software projects. Currently, it is been used to analyze big projects in the execution of several case studies.

TEDMA provides a way to experiment with different metrics required for technical debt management. Currently, there are several approaches to measure technical debt, but it is not clear in literature which ones should be used or how to use them together [126]. TEDMA helps to analyze how these different techniques work in specific projects. This is important for software developers because it allows them to analyze which of all the available tools that calculate metrics for technical debt management are the most useful for their projects. Additionally, TEDMA stores all the information in databases that can be exploited externally by other tools of the organizations.

The expected evolution of TEDMA will make it useful for software development industry. Using TEDMA, organizations will be able to manage the technical debt of their projects by selecting the tools and indicators that are the most important for them.

6.5 Conclusions

TEDMA provides the required support to use the technical debt management elements defined in Chapter 4 in real projects. In fact, TEDMA creates a platform to experiment with technical debt tools and strategies. TEDMA is thought to integrate third-party tools. Thanks to TEDMA, a case study, which is described in Chapter 7, could be performed in a big software project. In the future, more empirical studies will be performed using TEDMA.

This chapter complements the Chapter 5 to accomplish the Contribution 3: Identification of tools and strategies that support the elements identified in Contribution 2 and the lacks in this support. The analysis of the support of tools and strategies for the technical debt management elements indicated the necessity of integrating several different techniques. TEDMA responds to that necessity by providing a tool that is designed to integrate third-party tools. In fact, TEDMA facilitates the usage of the technical debt management elements in real projects.

Part V

Identification of how software internal quality increases the customer value

Chapter 7

Decision-Making Support Model

The goal of this chapter is to put into practice the technical debt management element framework defined in Chapter 4. To do that a model for technical debt management that takes into account time-to-market has been defined and used in a case study.

This chapter is an excerpt from the following papers:

- Carlos Fernández-Sánchez, Juan Garbajosa, Jessica Díaz, Jennifer Pérez, The Relationship between Technical Debt Management and Time-to-Market: An Exploratory Case Study, Submitted to IEEE Transaction on Software Engineering.

7.1 Introduction

The last main contribution of this thesis, Contribution 4, is *Definition of a model for making decisions on software evolution using the elements identified in Contribution 2 and that integrates tools and strategies identified in Contribution 3.* Previous chapters identified methods for technical debt management. The identified methods and strategies for technical debt management do not consider time-to-market; this situation can lead to wrong decisions. The present chapter takes advantage of the outputs of the previous contributions to define a model for technical debt management that considers time-to-market. This chapter uses the framework defined in Chapter 4 to define the model, the methods and strategies identified in Chapter 5 to implement different elements of technical debt management, and TEDMA tool, which is described in Chapter 6, to obtain data to apply the model in a case study using a large software project. Therefore, this chapter presents the model proposed and a case study where the model was used.

The remainder of the chapter is organized as follows: Section 7.2 will present the theoretical foundation of the study. Section 7.3 will present the model proposed in this paper for technical debt management when considering time-to-market. Section 7.4 will describe the case study design. Section 7.5 will present the exploratory case study execution. Section 7.6 will discuss the outcomes that were found out in the case study. Section 7.7 will discuss the limitations of this study. Section 7.8 will discuss previous related studies. Finally, conclusions and recommendations for future work will be presented in Section 7.9.

7.2 Background

In this section, the theoretical foundation of this chapter is presented. According to Runeson and Höst [94], in a case study, it is necessary to define the frame of reference of the study to make the context of the case study's research clear and to help both those conducting the research and those reviewing the results of it.

7.2.1 Technical debt in the context of software evolution

As was defined in previous chapters of this thesis, the term technical debt is a metaphor that refers to the consequences of weak software development. Technical debt management consists of identifying the sources of the extra costs of software maintenance and evolution, and determining whether it is profitable to invest efforts into improving a software system [114]. A technical debt item is a weakness in software that can cause internal quality problems [64]. In this chapter, technical debt is defined in terms of principal and interest [19]. The principal of a technical debt item is the cost of fixing the weakness associated with the technical debt item. The interest of a technical debt item is the extra cost of software maintenance and evolution that is caused by a non-optimal system structure and realization. The Chapter 4 identified and described technical debt management *elements*, some of which are principal and interest.

Lehman and Ramil [67] already pointed out that most of the software is produced because of a continual, subsequent evolution. Continuous software engineering has extended and formalized this way of understanding software development [41, 15]. Software typically is delivered as a sequence of releases. According to [104], more than 75% of organizations deliver software updates at least once a month and 44.7% do so monthly. Therefore, software is more often developed under an evolution paradigm.

It is in the context of this continuous evolution where technical debt should be managed. To express the concept of evolution, Schmid [98] used the concepts of external and internal quality. Following Schmid [98], the external quality of a software product is the accumulation of any observable quality of the product at run time (e.g., a functionality or a level of performance); the internal quality of a software product is the accumulation of any non-observable quality of the product at run time (e.g., code quality, complexity of the code, or number of source files). Schmid [98] stated that although an evolution step can describe any form of change in the external quality of the product, internal quality changes alone are not evolution steps (e.g., refactoring). Following also Schmid [98], within this paper, the evolution of a product is understood as any form of change that leads to an observable effect at the systems level.

Although there is a large number of models for release planning in the literature, these models have not yet reached the maturity required by industrial contexts and do not yet deal with possible changes in internal quality together with the implementation of new features [96, 110, 7].

7.2.2 A model for studying the trade-off between time-to-market and product performance

For the purpose of this thesis, a product development process model that could consider time-to-market was required. In particular, it is the model described by Cohen et al. [24] for new product development, the term used in innovation management for the development of products, because a model that was specifically produced for software development could not be found. The software evolution paradigm fits very well with

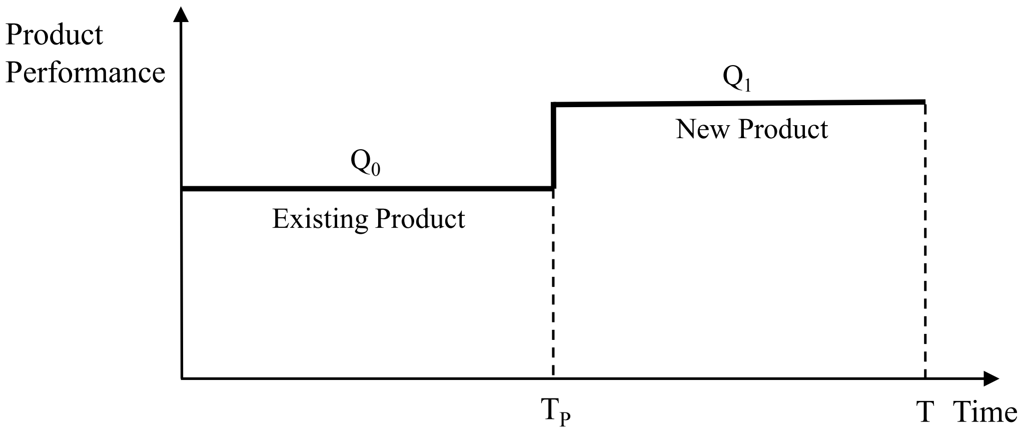


Figure 7.1: The performance of product in the marketplace over time (source [24])

the new product development paradigm: a release (or a set of releases) maps to a new version of the product under development; in fact, more often, software release management and new product development management are considered together, as was recently the case in [70].

Engineering management has considered time-to-market a key issue for years now. Cohen, Eliasberg, and Ho published a seminal modeling framework for studying the trade-off between time-to-market and product performance in new product development [24]. This modeling framework was later extended in [25] to include in the trade-off the level of resource intensity employed during the development process.

Cohen et al.'s modeling framework was defined by a set of formulas for a context depicted in Figure 7.1. This context describes a scenario in which a new product is planned so that it could replace an existing product in such a way that the product *performance* is increased by including new characteristics (see Figure 7.1). In this context, *performance* means value to customers. In Figure 7.1, T_P is the launching time of the new product, and T is the estimated end of life of the new product. Therefore, T_P and T define a time frame in which the analysis can be performed.

The performance of the new product at its launching time, $Q(T_P)$, is described as follows:

$$Q(T_P) = Q_1 = Q_0 + K * L^{\alpha} * T_P, \tag{7.1}$$

where

T_P = launching time of the new product

Q_1 = new product performance

Q_0 = previous product version performance

L = size of the development team

α = resource productivity parameter (there are diminishing returns to resource input, thus $0 < \alpha < 1$)

K = is the constant of proportionality for the speed of performance improvement. It is proportional to the level of capital investment in the development of technology.

Equation 7.1 models the performance (value to customers) of a new product that replaces an old version of the same product. The performance of the new product depends on the performance of the old version (Q_0), the effort spent (L^α and T_P), and the capacity to add value to the product by the company (K).

The total development cost of the new product, $TC(T_P)$, is described as follows:

$$TC(T_P) = W * L * T_P, \tag{7.2}$$

where

W = labor cost

L = size of the development team

T_P = launching time of the new product

The total net revenues of the new product, $TR(T_P)$, is described as follows:

$$TR(T_P) = M * m_0 * \frac{Q_0}{Q_0 + Q_c} * T_P + M * m_1 * \frac{Q_1}{Q_1 + Q_c} * (T - T_P), \tag{7.3}$$

where

M = product category demand rate

m_0 = margin of the existing product

m_1 = margin of the new product

Q_0 = previous product version performance

Q_1 = new product performance

Q_c = competitive product performance

T_P = launching time of the new product

T = end of the time window

Thus, the firm's cumulative profit, $T\Pi(T_P)$, is described as follows:

$$T\Pi(T_P) = TR(T_P) - TC(T_P), \tag{7.4}$$

where

$TR(T_P)$ = total net revenues

$TC(T_P)$ = total development cost

The framework has the capacity to describe more complex scenarios. Nevertheless, Cohen et al.'s modeling framework, as described in this paper, is sufficient for our objectives: it can be advantageously used to perform several analyses, including optimal time-to-market and product performance. Moreover, these analyses also consider market factors such as competition and margin of the product.

7.3 How to use Cohen's et al. model in software development for managing technical debt

Cohen et al.'s [24] modeling framework is focused on product markets characterized by a short and fixed window of opportunity, a high rate of product obsolescence, and customers who understand and respond to product performance improvements. This market is similar to the software market in which (i) software products must be developed in a time frame to be competitive with regard to competitors, (ii) software products must be replaced by new versions in very short time periods, and (iii) customers usually choose the products with the best functionality for their necessities, as explained in [110]. Cohen et al.'s modeling framework can be used with a constant or variable size of the development team [25]. Here, a constant size was used, which does not imply that the variable approach cannot be used as well. The constant size can perfectly describe a real situation in a product development scenario.

The original modeling framework presented by Cohen et al. [24] allows for modeling complex scenarios in which the product development can be split into several steps (e.g., design and development steps). Nevertheless, Cohen et al. developed their framework in [24] using two steps (design and process); in our case, we follow a very similar scheme. Within this paper, the development of a new product will be modeled in two sequential stages: (1) removing technical debt and (2) developing new features to include them into the product. Once the development is finished, a new release is produced. Because refactoring is the most used approach to repay technical debt [69], in this study, removing technical debt is considered a refactoring process. For the purpose of this study, refactoring will be performed before development to include new features. The stages are defined by T_R, the time for removing technical debt, and by T_P, the time for launching the new product. Therefore, the time frame is described by T_P, T_R, and T, where T is the expected retirement of the new release of the product. The context of this model follows the scenario described in Figure 7.2, in which the current release will be replaced by a new one.

Following the definition of technical debt provided in Section 7.2.1, removing technical debt is implemented as a refactoring process in which the internal quality of the product is improved without adding external quality. Thus, the product performance

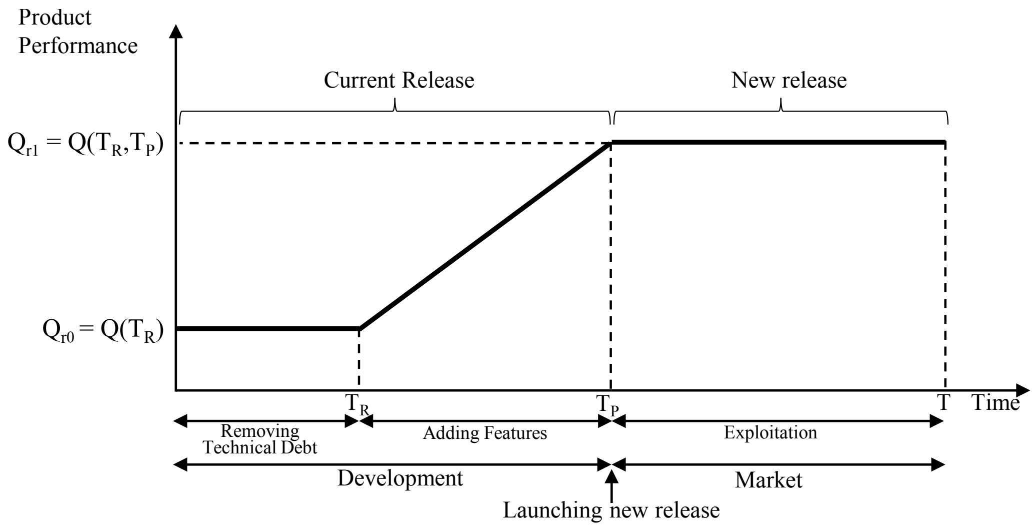


Figure 7.2: The performance of product in the marketplace over time (source [24])

(value for the customers) is not increased by removing technical debt, but it is considered a capital *investment* in the development of technology, that is, an improvement in the production capacity by *investing* in the internal quality. Therefore, we consider the effort in removing technical debt as a capital investment in the development of technology for the product; therefore, it will directly impact the K constant in Equation 7.1 [1].

The model for managing technical debt that takes into account time-to-market is defined by a set of formulas derived from Equations 7.1, 7.2, 7.3, and 7.4. The only change was that the development process was split into two steps: (1) removing technical debt and (2) development for including new features. The formulas were adapted following the same approach used by Cohen et al. [24] so that the model could be used as a development process split into two stages.

Using Equation 7.1, the performance of removing technical debt, $Q(T_R)$, can be modeled as follows:

$$Q(T_R) = Q_{r_0}, \tag{7.5}$$

where

T_R = time at the end of the removing technical debt stage

Q_{r_0} = previous product version performance

[1]Similarly, other investments could be made, for example, acquiring new development tools or workstations. Our model is only focused on refactoring the code. Other types of investments are left for future research.

This equation means that because removing technical debt does not improve the external quality of the product $Q(T_R)$, it does not increase the performance of Q_{r_0}[1].

Using Equation 7.1, the *performance of the new product* at its launching time, $Q(T_P)$, is described as follows:

$$Q(T_R, T_P) = Q_{r_1} = Q_{r_0} + K * L^{\alpha} * (T_P - T_R), \tag{7.6}$$

where

T_P = launching time of the new product

T_R = time at the end of removing technical debt stage

Q_{r_1} = new product performance

Q_{r_0} = previous product version performance

L = size of the development team

α = resource productivity parameter (there are diminishing returns to resource input, thus $0 < \alpha < 1$)

K = the constant of proportionality for the speed of performance improvement. It is proportional to the level of capital investment in the development of technology.

Using Equation 7.2, the *total development cost* of the new product, $TC(T_R, T_P)$, is described as follows:

$$TC(T_R, T_P) = W * L * T_R + W * L * (T_P - T_R), \tag{7.7}$$

where

W = labor cost

L = size of the development team

T_P = launching time of the new product

T_R = time at the end of removing technical debt stage

Using Equation 7.3, the *total net revenues*, $TR(T_R, T_P)$, is described as follows:

$$TR(T_R, T_P) = M * m_0 * \frac{Q_0}{Q_0 + Q_c} * T_P + M * m_1 * \frac{Q_1(T_R, T_P)}{Q_1(T_R, T_P) + Q_c} * (T - T_P), \tag{7.8}$$

where

M = product category demand rate

m_0 = margin of the existing product

m_1 = margin of the new product

[1]It might be possible that refactoring the software would generate problems that decrease the external quality (e.g., adding bugs), in such cases $Q(T_R) <= Q_{r_0}$. Within this study, we have not considered such situations.

Q_{r_0} = previous product version performance
Q_{r_1} = new product performance
Q_c = competitive product performance
T_P = launching time of the new product
T_R = time at the end of removing technical debt stage
T = end of the time window

Using Equation 7.4, the firm's cumulative profit, $T\Pi(T_R, T_P)$, is described as follows:

$$T\Pi(T_R, T_P) = TR(T_R, T_P) - TC(T_R, T_P), \tag{7.9}$$

where
$TR(T_R, T_P)$ = total net revenues
$TC(T_R, T_P)$ = total development cost

The *primary decision* consists of choosing the values for T_R and T_P to obtain the *maximum profit*. Other decisions about team size or time window can be made. The model allows for making trade-offs between time-to-market and performance (value for the customers), taking into account other market information such as competitive performance and margin of the new product. Therefore, this model can be used to support technical debt management, including technical and business points of view.

7.4 Case study design and planning

In this section, we present the case study in which the previously defined model was used. We followed the guide of Runeson and Höst [94] to conduct this exploratory case study. The next subsections are organized following their guide.

7.4.1 Objectives

The goal of this study was to explore technical debt management in large software projects while considering time-to-market to discover the constraints and limits that should be considered. Because there are not enough industrial relevant studies that use all the required elements for technical debt management [39], we performed an exploratory case study to seek new insights and generate ideas and hypotheses for new research. Specifically, we are interested in technical debt management in projects with low data availability and in the study of the limitations that can arise in such kind of situations and how these limitations can be addressed.

Examples of project contexts in which we were interested are open source projects that provide access to the source code but do not provide extra documentation about the software architecture, design decisions, and so forth. This information is difficult to be obtained, especially if the original team left the project. Other examples are legacy projects without documentation and without access to the original development team.

It is not the goal of this study to discuss which methods perform better than others in measuring or estimating technical debt. As explained in Section 7.5, different methods for technical debt estimation that were previously used in other studies were used in the current case study to use the model defined in this paper.

7.4.2 Rationale

Fernández-Sánchez et al. [39] studied the elements required to manage technical debt. One result of this study is that time-to-market is not used widely in technical debt management [39], even though it is one of the most widely mentioned causes of technical debt [114]. The case study described in this paper was performed to explore how to integrate time-to-market in technical debt management. Therefore, we studied how to use the currently defined modeling framework for product management (described in Section 7.2.2), and took into account time-to-market, in software development. Specifically, we used such a modeling framework in technical debt management. Additionally, we followed the framework described in [39] as a guide for technical debt management.

7.4.3 Case and subject selection

Because of the wide use of the *Java* language for constructing systems[1] and because the experience of the authors was broader in *Java* programming than in other languages, the project used to produce this exploratory study was a *Java* project. This was done so that in case code inspection was needed, it could be better performed. Java was also selected because the authors were, at the time of producing the study, more familiar with the tools to analyze *Java* projects. Working with known tools reduces the risk attached to working with unfamiliar technologies, such as a misunderstanding of the tools' outputs.

An open source project was selected so that access to the project source code repository could be granted. The authors agreed that the selected project should be large enough to incorporate complex scenarios. Following these considerations, the project selected was the *Apache Log4j 2* [2]. This project has more than 1,000 files and more than two years of development after its first release.

[1] http://spectrum.ieee.org/computing/software/the-2016-top-programming-languages checked March 18, 2017

[2] https://logging.apache.org/log4j/2.0/ checked March 15, 2017

Apache Log4j 2 is a well-known project by the *Java* community, and it has been used previously in several research studies, for instance in [88, 77]. When conducting this case study, *Apache Log4j 2's* development team was working on release 2.7. The project had evolved over nearly two and a half years, and it contained almost 1,500 Java files. In this way, as part of this exploratory study, the analysis of the project was performed by the researchers, who did not have knowledge about how the project had been developed. The only available data sources were the source code repository[1] and the ticket repository used by *Apache Log4j 2*.

How the authors of this paper face the exploratory study is similar to the scenario in which development teams have to start a project, not having much information about the existing code except for the code itself. Probably, the *Apache Log4j 2* team will have such information internally, and it could even be shared publicly, but for the sake of the study, we decided to only use the information provided by the *Apache Log4j 2* track tool, *JIRA*[2]

7.4.4 Theoretical frame of reference

The theoretical framework of this study was formed by the available studies about technical debt that were analyzed in [39], [6], [9], [69], and [114]. In the literature, several techniques had been used to implement specific parts of technical debt management. In this study, some of these techniques were adapted to be used in the analyzed project. In the context of product development and time-to-market, Cohen et al.'s studies [24, 25] were used as the reference of how time-to-market is considered in management science. More detail about the theoretical background of the study is provided in Section 7.2.

7.4.5 Methods, data collection, and selection of data

To analyze the data of the *Apache Log4j 2* project, we used releases as points of stable states of the project. For this, the information was extracted from the *JIRA* track tool. We obtained the data revision by revision and accumulated them into releases. To do this, we developed a tool for mining source code repositories. This tool for extracting the information can track all the renames of the files over the entire history of the project, so no data were missed by changes in the name or by moving a file to a different directory. Once the project is mined, it is possible to follow the history of each file. Because we were focused on the project's evolution, we did not include data about the first release of the project (release 2.0).

[1]https://github.com/apache/logging-log4j2

[2]https://issues.apache.org/jira/browse/LOG4J2/, checked March 15, 2017

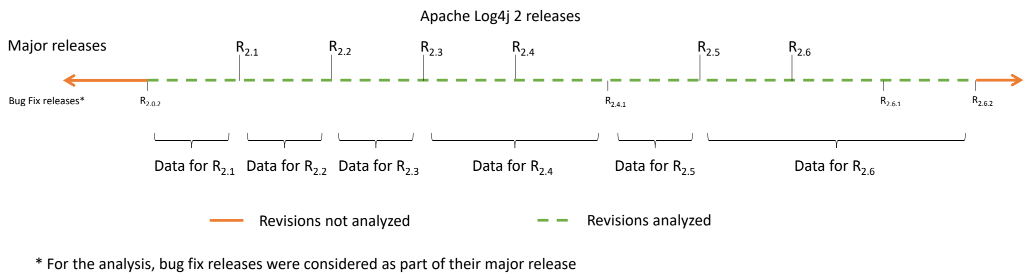


Figure 7.3: *Apache Log4j 2* releases analyzed in this study

We extracted data from six consecutive major releases: 2.1, 2.2, 2.3, 2.4, 2.5, and 2.6. To avoid a distortion of the data with the different granularity of the releases, that is, releases focused on fixing bugs and adding small improvements and releases adding new functionalities, bug fix releases were considered a part of their major release. This does not avoid that different major releases have different sizes (one release can imply more functionality and effort than others), but at least we avoided comparing known small releases with known large releases. Therefore, we analyzed the evolution of the project from release 2.0.2 (we considered it a part of the release 2.0) until release 2.6.2 (considered a part of the release 2.6). Figure 7.3 shows the releases used in the analysis. At this point, it is necessary to highlight that several releases can be developed in a parallel way. For example, in Figure 7.4 and the example with two branches, releases r_1 and r_2 have developments in a parallel way, specifically, revisions rev_3 and rev_4 are developed in a parallel way. This was solved as it is shown in Figure 7.4. The main consideration was that the method that accumulates revisions on releases was guided by the branches and merges involved in the history of the releases, not by the age of the revisions. This guarantees that the changes included in a revision were only in one release and that they were accumulated in the correct release.

7.4.6 Case study protocol

Data extraction was performed in an automatic way; therefore, the procedures for data extraction were documented within the source code of the tool developed for mining source code repositories. Furthermore, the tool used for data extraction is an excellent way for replication of the data extraction. The data extracted were stored in a database in which standard query languages can be used to extract or consult data. All the data analyses were performed using R. Consequently, all the process in the analysis were written in R functions that allowed the replication of the analysis. This facilitated the

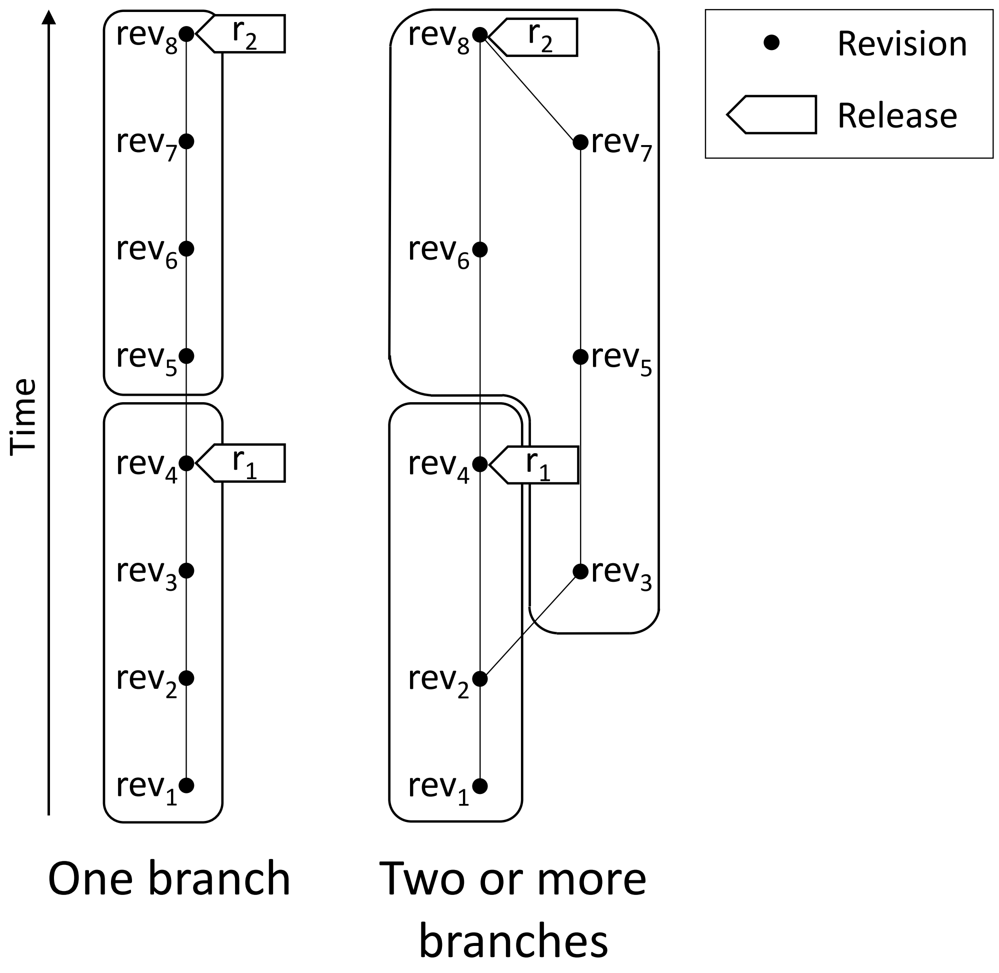


Figure 7.4: Examples of how revisions are considered to be in a release

revision of the analysis by all the researchers involved.

7.4.7 Ethical considerations

We are not describing the quality of the project analyzed or the quality of the development team. We are just analyzing the impact of some anti-patterns (common to most software projects) in the technical debt of the project. Furthermore, these patterns could be known by the development team, and they could be not removed by informed design decisions. We decided to use only source code repository and ticket repository as data inputs. This does not mean that the analyzed project does not have more documentation sources.

7.5 Case study execution

We used a framework that identifies the elements required for technical debt management [39] (see Figure 7.5). This framework identifies the minimum data required for basic technical debt decision making. We did not integrate the technical debt management process into the software development process. Because we were external to the project team, our approach was more similar to "computer forensics" of the project. To integrate this analysis with a software development process, the activities for technical debt management identified by Li et al. [69] might be used as a guide. Based on the framework for technical debt management (see Figure 7.5), the following subsections report some of the studied technical debt elements. This includes the definition of a number of indicators for some of the elements (Sections 7.5.1, 7.5.2 and 7.5.3.), the analysis of the technical debt impact element in the function of, though not exclusively, the time-to-market element (Section 7.5.4), and finally the analysis of the results through a scenario analysis (Section 7.5.5).

Over the execution of the case study, because we decided to use only the source code repository and the track tool, there were several adaptations of the model described in Section 7.3. All the adaptations were motivated to adapt the model to a context with few information sources. The adaptations are described in the following subsections.

In the next subsections, the required information for technical debt management and how it was obtained are explained.

7.5.1 Technical debt items indicators

Technical debt item identification (see Figure 7.5 element E1), was the first element tackled. This required, first, to analyze this element in the context of the study and,

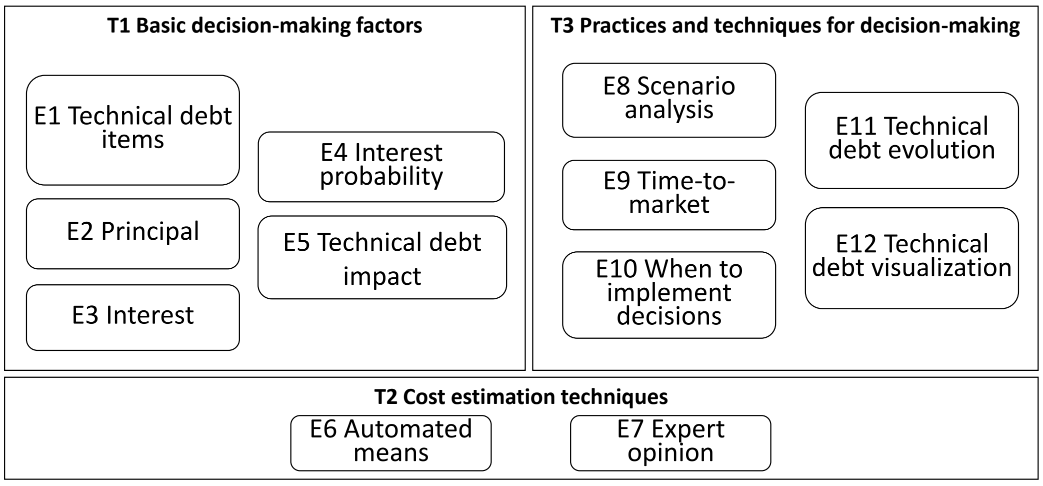

T1 Basic decision-making factors
E1 Technical debt items
E2 Principal
E3 Interest
E4 Interest probability
E5 Technical debt impact
T3 Practices and techniques for decision-making
E8 Scenario analysis
E9 Time-to-market
E10 When to implement decisions
E11 Technical debt evolution
E12 Technical debt visualization
T2 Cost estimation techniques
E6 Automated means
E7 Expert opinion

Figure 7.5: Identified elements for technical debt management. Source [39].

second, to define a number indicators necessary to estimate the technical debt of the *Apache Log4j 2* project. A technical debt item is a weakness in software that can cause internal quality problems. In the present study, we used the term technical debt item to refer to a file with technical debt indicators. There are several approaches to identify technical debt items [39]. It is not the goal of this paper to discuss which methods perform better than others. Furthermore, as studied in [126], it seems that different methods could point to different technical debt items. So these different methods are, in some way, complementary. In any case, it was not the goal of the study to compare different technical debt identification techniques. Thus, we chose one indicator frequently used in other studies because there is no clear evidence of what technical debt indicators, among all the ones studied, should be used in technical debt management. In the literature, several code smells and anti-patterns that cause maintainability issues have been used as indicators of technical debt [126, 103, 73, 101]. Therefore, we decided to use the code smells that the tool PMD [85] detects as potential problems in maintenance. This includes some code smells as the God Class that has been widely used in technical debt management. Nevertheless, the use of some code smells in some projects does not guarantee that those code smells will be valid indicators in other projects. This situation made it necessary to select which code smells were the most accurate for identifying technical debt items in the *Logj4 2* project. A way to analyze if technical debt indicators are identifying items with technical debt is comparing the cost of changing files with the technical debt potential indicators and the cost of changing files without such indicators [126]. Based on this, we considered that an indicator of technical debt was valid if files with such an indicator required more effort in the evolution of the software than files without the technical debt indicator. Thus, it became necessary to define a condition that the set of valid indicators should fulfill, as follows:

Condition 1 *Files with technical debt indicators have higher values in effort indicators than files without technical debt indicators.*

This condition requires an effort metric that can be estimated when the condition is applied to verify if a technical debt indicator is valid. If we have access to the Git repository and if the website does not provide any effort estimation, we had to find a workaround, similar to what happens in industrial projects whenever effort data are not carefully collected, not an uncommon situation, and effort data are not associated at file level [59]. To identify which technical debt indicators fulfill Condition 1, we followed a similar strategy to the one followed by Kazman et al. in [59] and introduced two typical effort metrics used in the literature for technical debt management: changed lines and change probability [126]. We defined these two indicators as follows:

Definition 1 *ChP_f is the change probability of file f. It is the probability that file f will change in the next release. It is calculated by dividing the number of releases in which the file has changed by the total number of releases analyzed in which the file f exists. As a probability, it will take values between 0 and 1.*

Definition 2 *$ESCh_f$ is the expected size of change for file f. It is the expected number of lines that will change in the next release. It is calculated as the average number of lines changed (added, removed, or modified) in file f over the releases analyzed.*

Combining the previous two metrics, we can define the expected change effort as follows:,

Definition 3 *$EChE_f$ is the expected change effort of file f, and it is the result of multiplying the $ESCh_f$ and ChP_f. It represents the expected effort in lines that will be changed in file f over the next release considering both the size and probability of the change.*

We used $EChE_f$ because it includes the other two indicators.

At this point, we used these metrics to determine if the *Apache Log4j 2* technical debt indicators satisfy Condition 1. Table 7.1 shows the values of ChP, $ESCh$, and $EChE$ measured in the files of the *Apache Logj4 2* project and accumulated by technical

Table 7.1: Expected size of change, change probability, and expected change effort accumulated by technical debt indicators (indicator definitions in [85]). *ESCh*: expected size of change in lines of code; *ChP*: change probability; *EChE*: expected change effort in lines of code; *L*: average lines of code per file.

TD indicator	*ESCh*	*ChP*	*EChE*	*L*	Files
with Excessive Public Count	575.86	0.75	471.53	919.72	12
with Excessive Parameter List	385.10	0.84	378.25	451.97	53
with God Class	294.94	0.77	264.88	553.56	51
with N Path Complexity	259.85	0.66	248.21	382.59	62
with Excessive Method Length	197.55	0.68	184.88	577.22	18
with Too Many Methods	203.39	0.69	179.29	343.60	137
with ANY TD Indicator	145.60	0.65	128.31	300.26	253
with Cyclomatic Complexity	115.15	0.57	104.13	320.48	99
ALL FILES	46.14	0.45	38.25	119.70	1456
without TD Indicators	31.73	0.43	26.40	87.97	1255
with Too Many Fields	-	-	-	-	0

debt indicators. Because the values were accumulated, they were calculated as average values. For example, the *ChP* of files with a God Class anti-pattern was calculated as the average of the change probability of all the files with God Class. No file with the Too Many Fields anti-pattern was found. Table 7.1 also includes rows for the values of all the files in the *Apache Logj4 2* project, files with any TD indicator, and files without technical debt indicators. The table is ordered by *EChE*. As can be seen, files with technical debt indicators had a bigger *EChE* (¿= 104.13 lines of code) than files without technical debt indicators (26.40 lines of code). Thus, it could be concluded that the indicators satisfied Condition 1 and that these technical debt indicators seemed valid to be used in the context of the *Logj4 2* project.

In Table 7.1, *ESCh*, *ChP*, and *EChE* are based on the number of changes in the files and the size of changes in the files. There is a controversy about if metrics should be normalized by the size of the files [126, 124]. As highlighted in [126], to normalize by the number of lines would make sense if there was a linear dependency between the metrics used and the size of the files. Table 7.2 shows the Pearson coefficient for the variables used in Table 7.1 and the size of the files. Once the coefficient is known, it would be possible to interpret a linear relationship. To analyze this possible relationship, we used linear regression. We created models using the variable *file lines*

Table 7.2: Pearson correlation coefficient of size of files in lines with expected changed lines, Change Probability, and Expected effort.

	Pearson correlation
Expected changed lines	0.6805006
Change probability	0.3190039
Expected effort	0.6444562

as a predictor for the other variables. The first row of Figure 7.6 shows the scatter plots and the linear regression. The second row shows the residual of the regression. As can be seen, the residuals show patterns instead of showing random values; therefore, the regression models were invalid for prediction. This means that only the number of lines could not explain the other variables, that is, there were other variables that also influenced the size of change, probability of change, and, consequently, expected effort. Therefore, the decision of not normalizing using the size of files was made.

We observed that some combinations of technical debt indicators pointed to bigger expected change effort. For example, in Table 7.1, it can be seen that the 51 files with God Class had an average of 264.88 *EChE*, and the 137 files with Too Many Methods had an average of 179.29 *EChE*. Making the intersection, the files with both anti-patterns (29 files) had an average of 341.78 *EChE*. That is, the subset of files with the two indicators seemed to have larger and more frequent changes than files with only one of the indicators. This behavior happened frequently but not with all indicators and combinations. Therefore, it is necessary to study this finding to extract deeper conclusions. But it seems that it could be possible to identify patterns of indicators that would point to the most problematic files in terms of technical debt. The most extreme case was the combination of God Class, Excessive Parameter List, and Excessive Public Count. Files with this combination of anti-patterns had 4752.67 *EChE*, more than ten times the expected change effort of all the files with Excessive Public Count (the individual indicator with the biggest expected change effort).

7.5.2 Principal indicators

The principal was the second element to be addressed (see Figure 7.5 element E2). It is the effort required to remove technical debt from the files with technical debt items. In software engineering, the estimation of a project's range of effort can be determined using methods such as expert judgment or analogy, and as long as historical size and effort data exist, using a calibrated estimation model is possible [106, page 7.6].

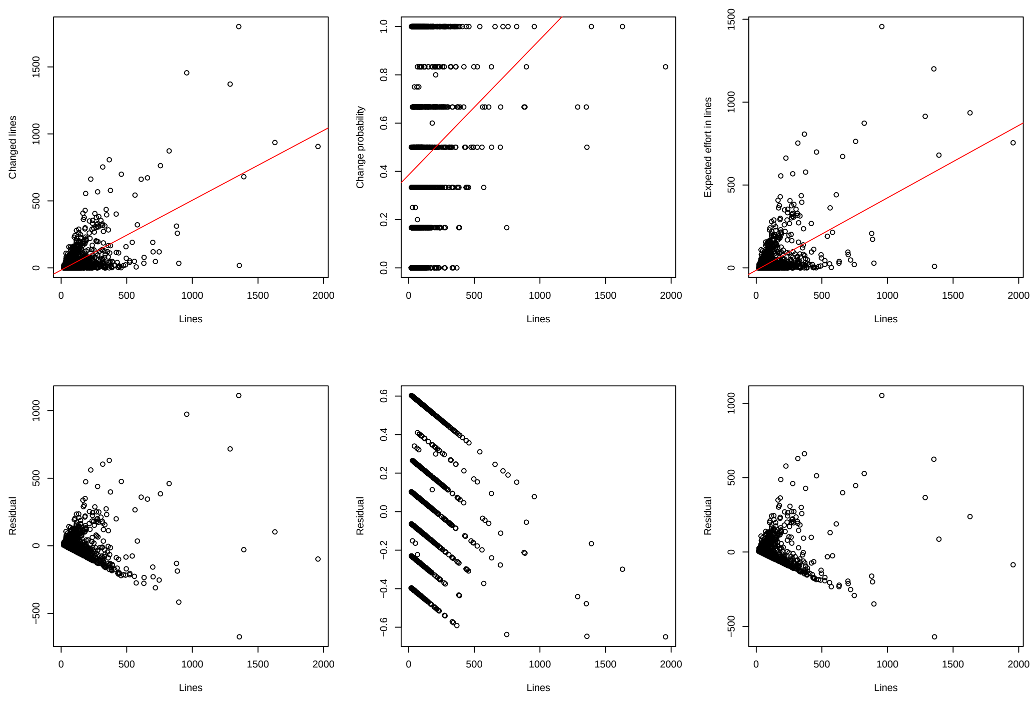


Figure 7.6: Linear regression models for file lines, changed lines, change probability, and expected effort.

If the team analyzing technical debt has deep knowledge about the project, accurate estimations about development effort can be provided. But as discussed by Schmid, cost estimation generates large uncertainty in software engineering in general and in technical debt management in particular [98]. As the knowledge on the *Apache Logj4 2* project was not available, to perform estimations, a strategy aligned with what followed in Section 7.5.1 was applied. In this case, the assumption was that refactoring a file with technical debt implied a change in all its lines. This was a conservative assumption because it is probable that many refactorings could be done without changing the file completely.

7.5.3 Interest and interest probability indicators

Interest and interest probability were the two following elements to be considered (see Figure 7.5 elements E3 and E4). Interest is the extra effort that is used in modifying the files with technical debt. In the literature, expected change size [59] was used as interest indicator. This estimation can be obtained by analyzing the evolution of a project repository to see all the changes over the history of a project. The same approach was used for estimating the expected size of the change, represented by $ESCh$. That is, for a file with technical debt and with an expected size of change ($ESCh$), removing its technical debt should return a decrease in its expected size of change ($ESCh$). In Section 7.5.1, $ESCh$ was estimated as the expected number of lines that a file will change in the next release.

It is necessary to highlight that in technical debt, the interest is not always paid. The probability of paying the interest depends on the probability of the occurrence of future events [64]. Because we focused on evolution, this probability can be estimated as the probability of change of the technical debt items. The change proneness or change probability has also been used in the literature as a metric of the interest of technical debt items [126]. As for interest, it can be estimated from historical data. As shown in Section 7.5.1, the change probability of a file ChP_f is the probability that the file f will change in the next release. It was calculated by dividing the number of releases in which the file f had been modified by the total number of releases analyzed in which the file f existed. To combine the interest and interest probability, we used the expected change effort $EChE$, as described in Definition 3.

Therefore, following the same approach that Schmid followed in [98] to define technical debt, we defined the interest in terms of variations of interest between different alternatives.

Definition 4 *The change of interest of a file with a technical debt indicator is* $\Delta I_f = EChE_{f,r} - EChE_{f',r}$*,* $EChE_{f,r}$ *is the expected change effort of file* f *for the release* r*.* f' *is an alternative of* f*, being* $f' = refactor(f)$*.*

To estimate the expected change effort of the alternative f', expressed by $EChE_{f',r}$, we followed the same approach used by Kazman et al. [59]. Kazman et al. estimated the expected change effort $EChE$ of a refactored file, making the assumption that after refactoring, the file will have an $EChE$ equal to the project's average. This estimation was conservative because project expected change effort $EChE$ average already includes files with technical debt indicators that inflated the average. Though, in this study, the assumption was that when a technical debt item is removed, the expected change effort $EChE$ is reduced to double of the project average, considering only the files with changes. Therefore, our assumption is even more conservative than that of Kazman et al. in [59].

In practice, the previous paragraphs lead to the conclusion that to properly manage technical debt, it is necessary to estimate the positive impact of a refactoring in future changes.

A final remark: to use proxies of effort, such as lines changed instead of "real measures" of effort, could introduce more uncertainty in the interest estimation. But even having measures of the real effort, they could not be precise because typically, they are recorded manually by developers. Furthermore, if they are not directly assigned to the technical debt items (in our case, at the file level) they are unsuitable for technical debt management [59].

7.5.4 Technical debt impact: introducing time-to-market

Technical debt impact was another element to address (see Figure 7.5 element E5). This element allowed us to relate technical debt and time-to-market. This relation was further elaborated in the element *Scenarios to make decisions* in Section 7.5.5.

In technical debt management, decisions consist of determining when it is profitable to remove technical debt items considering principal, interest, and interest probability. It is possible to consider more elements, which is the case with time-to-market.

With this objective of determining when it is profitable to remove technical debt when considering, though not exclusively, time-to-market, the model presented in Section 7.3 was refined to be expressed with the indicators defined in Sections 7.5.1, 7.5.2, and 7.5.3 as terms of the equations.

To use the performance of the new product (Q_{r_1}), described by Equation 7.6, the production capacity of the development team was needed. These data were not available. Therefore, the production capacity was approximated as the number of features per day of the project. Hence, Equation 7.6 can be simplified to the following:

$$Q(T_R, T_P) = Q_{r_1} = Q_{r_0} + \widehat{K * L^\alpha} * (T_P - T_R) \tag{7.10}$$

We estimated the team production capacity ($\widehat{K * L^\alpha}$) using the last project release. This was because the last release was the nearest in time when producing the study, and it represented the project production capacity at that moment. Revision $r_{2.6}$ lasted approximately six months, and it added 64 new features and improvements. Using an equivalence of 30 days per month, we estimated the production capacity of the project: $\widehat{K * L^\alpha} = \frac{64}{6*30} = 0.356 feature/day$.

To use the total development cost described by Equation 7.7, some approximations were also needed. The equation consists of two parts. The first one ($W * L * T_R$) represents the cost of refactoring, that is, the principal of the technical debt items fixed. For this part, we used the principal estimation, as was described in Section 7.5.2. The second part ($W * L * (T_P - T_R)$) represents the cost of adding new features to the product. We did not know the labor cost nor the size of the development team. Thus, as with the principal, the cost was approximated using effort instead of monetizing it. Therefore, as long as the changed lines of code were used as a proxy for effort, the changed lines were applied as an estimation of cost ($\widehat{W * L}$). Therefore, Equation 7.7 was refined to express total development cost in the study case with the following equation:

$$TC(T_R, T_P) = principal(T_R) + \widehat{W * L} * (T_P - T_R) \tag{7.11}$$

For revision $r_{2.6}$, with 76,118 changed lines, it was calculated as: $\widehat{W * L} = \frac{76118}{6*30} \approx 423 lines/day$

Finally, we refined Equation 7.8. The variable M refers to the demand rate of the market for the type of product under development; variables m_0 and m_1 represent the margin of the existent and the new product, respectively, and variable Q_c is the competitive performance. To measure or estimate the value of these variables implies to perform a market study and valuation of the business goals of the product. To obtain that estimations would have taken the study far from its objectives. Therefore, to represent the benefit, we used the performance of the new product, that is, $Q(T_R, T_P)$. Consequently, the total net revenues, described by Equation 7.8, was simplified as follows:

$$TR(T_R, T_P) = Q_1(T_R, T_P) \tag{7.12}$$

7.5.5 Scenarios to make decisions: rehearsing for time-to-market

This is the last element of the framework (see Figure 7.5 element E8) that was considered within this case study. In decision making, it is necessary to estimate the consequences of the decisions made about the system. By analyzing scenarios, managers can acquire information about the effects of the technical debt in the future. Developing this element helped us start to understand how technical debt management and time-to-market are related.

In this section, several scenarios are presented. As a first step, several scenarios in the context of the case study were defined; these scenarios are presented below. The scenarios were created based on the equations described in the previous sections. We defined a context based on the historical data of the project. Using this context, we defined different scenarios by changing some of the variables implied. The context to define the scenarios was set as follows:

- For principal estimations, we used the method described in Section 7.5.2.
- For estimating the interest and the interest avoided, we used the method described in Section 7.5.3.
- We used the estimations explained in Section 7.5.4, that is, $\widehat{K * L^{\alpha}} = 0.356 feature/day$ (team production capacity) and $\widehat{W * L} \approx 423 lines/day$ (development cost).
- The number features or improvements that were implemented in the product at the end of $r_{2.6}$ was 302. Therefore, we defined $Q_0 = 302$ (current performance of the product). This information was extracted from the *JIRA* track tool.

We chose the team production capacity and the development cost based on the last release analyzed because it was assumed that the context would be similar in the next release.

Using this information and equations described in Section 7.5.4, we analyzed the amount of effort to be invested in reducing technical debt that would yield a profit.

At this stage, the only missing task was to select which technical debt items would be removed. Once the items were identified, we needed to prioritize them. This was done by calculating a ratio of interest avoided (benefit) and the principal that should be paid (cost) if the item was removed. Depending on the time dedicated for refactoring, more or less technical debt items could be fixed and, more principal would be paid and, consequently, more interest would be avoided in the next release. Nevertheless, the more time there was for refactoring, the less time there would be for adding new features, and consequently, less value would be added for the customer.

7.5.5.1 Scenario 1

The first scenario, showed in Figure 7.7, shows a scenario where the time-to-market (T_P) was fixed to 180 days (a similar duration than release $r_{2.6}$). As can be seen, refactoring those files with a higher cost-benefit ratio has a positive impact on the final product performance ($Q_1(T_R, T_P)$), even in the subsequent release. But if the time for refactoring (T_R) is more than 25 days, the performance ($Q_1(T_R, T_P)$) of the subsequent release will be lower than if there was no refactoring. Therefore, if all the features planned were mandatory and we wanted to reduce more technical debt, this reduction on technical debt would have an impact on the time-to-market. In this situation, it is necessary to make a decision about if to release all the planned features, extend the time-to-market, or continue accumulating technical debt.

7.5.5.2 Scenario 2

In this scenario, shown in Figure 7.8, we fixed the time-to-market (T_P) to 90 days (a similar duration to release $r_{2.3}$). In this case, because of the short duration of the release, any reduction on technical debt has a direct impact on the performance of the product ($Q_1(T_R, T_P)$). Therefore, in that case, to reduce technical debt implies to deliver fewer features, and consequently, less external quality. This scenario shows how time-to-market is a relevant technical debt antecedent, supporting the findings of Tom et al. [114]. Following the present scenario, if the product needs to be released in 90 days with the maximum number of features, no technical debt would be removed, and probably more would be created. If the analysis was extended to more releases instead of only one, the refactoring could be valuable for further releases, but accepting that to achieve the time-to-market (T_P), fewer features should be implemented in the first release.

7.5.5.3 Scenario 3

Thanks to the use of Cohen et al.'s model, we were able to analyze what the minimum value for T_P (time-to-market) to implement a fixed number of features is. Figure 7.9

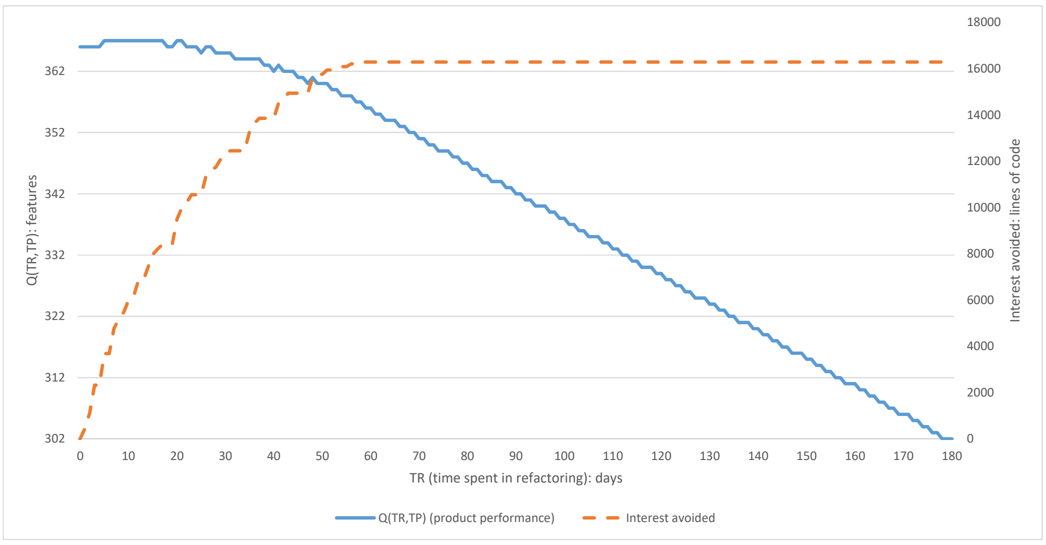


Figure 7.7: Scenario 1

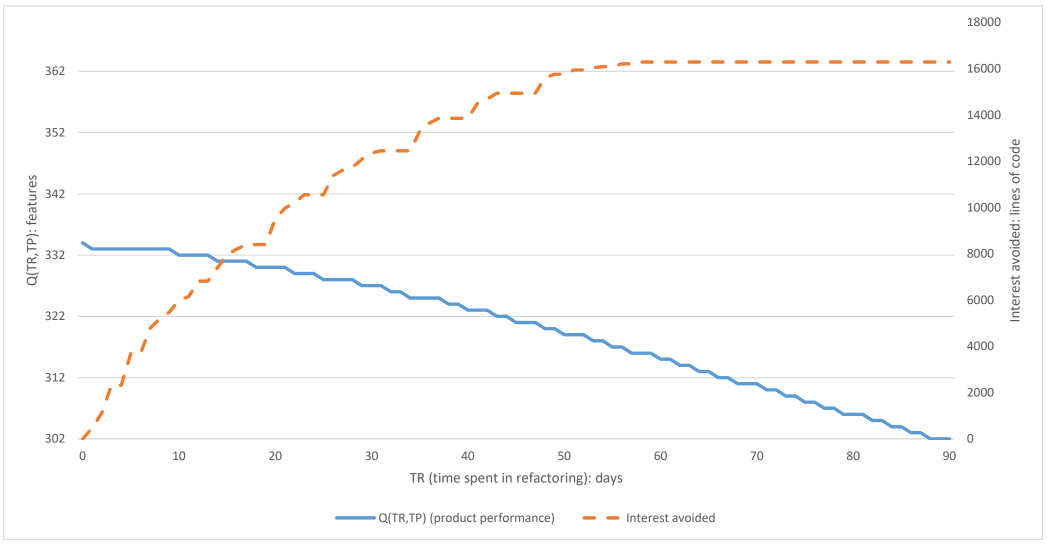


Figure 7.8: Scenario 2

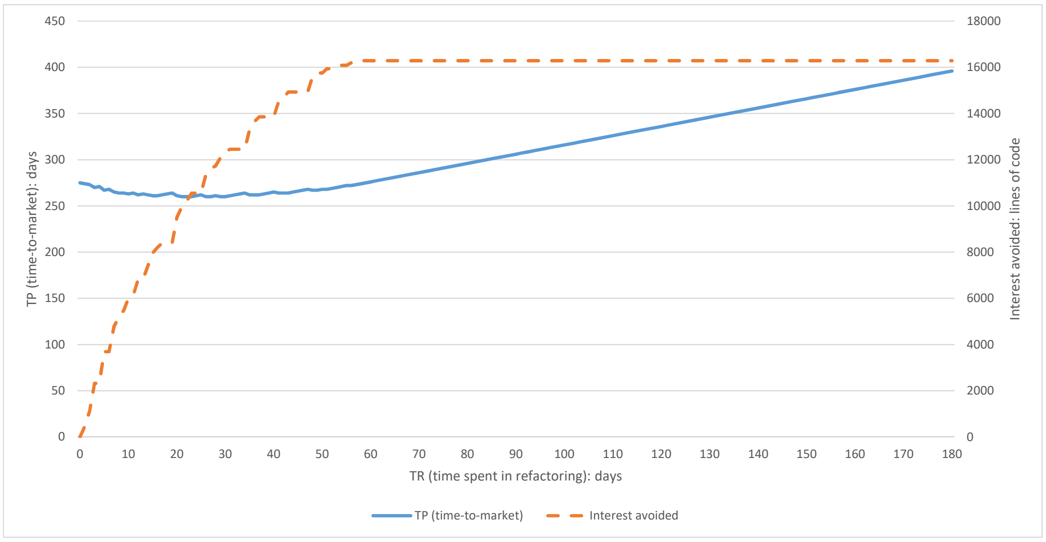


Figure 7.9: Scenario 3

shows the analysis, considering a goal of a product performance of 400 ($Q_1(T_R, T_P) = 400$) for the next release. As can be seen in Figure 7.9, the shorter time-to-market is to use 260 days for development, using 30 for refactoring (T_R). This allows one to choose the time-to-market based on the number of features to add, considering the benefit of removing the technical debt items that have bigger impact.

7.5.5.4 Scenario 4

This scenario, shown in Figure 7.10, is similar to Scenario 3, but in this case, the goal for product performance is 326 (i.e., to add 24 features as in release $r_{2.1}$). In this case, any time dedicated for refactoring (T_R) has a negative impact on the time-to-market (T_P).

Analyzing Scenario 2 and 4 together, it seems that short releases encourage technical debt to remain. This does not mean that it is better to have large releases. A long-term perspective is very important in technical debt management, short releases can lead to losing the long-term perspective, but if it is considered, for example, by analyzing several short releases in the roadmap of the product, this can be avoided. Furthermore,

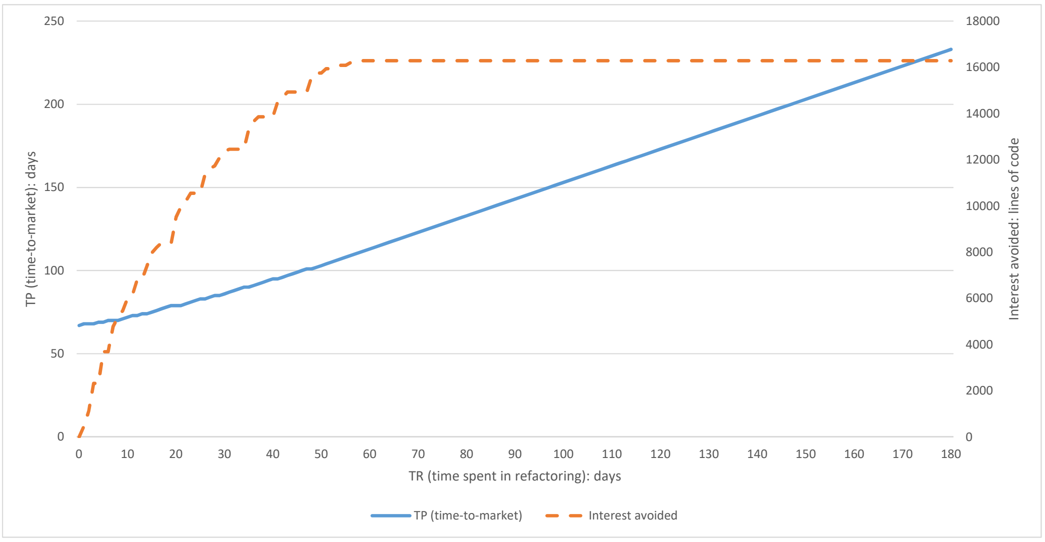


Figure 7.10: Scenario 4

analyzing the four scenarios presented here, it seems that to remove the technical debt items with more impact in each release could be enough to keep technical debt under control, even increasing the product performance on a similar level than without removing technical debt.

7.6 Findings

7.6.1 About the Cohen et al.'s model

Cohen et al.'s model [24, 25] has been adapted to be used in a software development context, specifically for technical debt management. The model has been used in a project where only code and a track issue repository were available. This was done deliberately because this is the most restrictive context where technical debt management can be done. To be adapted to this project, we had to make some simplifications that would not be required with more information. These simplifications helped the applicability of the model, but at the same time, this prevented the model from producing more detailed analysis of the technical debt of the software. In fact, to consider time-to-market in more detail, in technical debt management, several data are required: effort; value of new features; cost and benefit of refactoring that link technical debt management with software project management area; and the time frame for new products, performance of the product in the market, and performance of the competitive products that link with business management.

In a previous study [39], it was found that technical debt could be useful when taking into account different stakeholders with different points of view: business organizational management, engineering management, and engineering. As shown in Section 7.5.4, the model uses technical data, but also business data; this is evidence that technical debt management helps bring together different stakeholders' points of view.

The model is oriented toward trade-offs that have been useful for this study and have potential for future applications in the industry. Different versions of the product can be assimilated for new releases. It would be possible to consider different stages if we wanted to model more complex processes. However, this simple process model (adaptable to a specific team, product and, project constraint) allowed to reason about technical debt management and time-to-market. Therefore, technical debt management has been comfortably introduced as a process in software development.

The concept of product performance can be defined in terms of requirements prioritization and valuation, one of the drivers for release management [7].

7.6.2 Assumptions and approximations

The execution of the case study showed that it is necessary to perform many estimations for technical debt management. Those estimations are sources of uncertainty. The lack of data sources may make this situation worse. As discussed in Section 7.4.3, this is often the case when a team must start a project for which the team does not know either the code or the tools used to produce the code. As shown in the present study, many times, direct estimations are not available, and it is necessary to use proxies (see Section 7.5.4). This exacerbates the uncertainty of estimations and increases the risk of making decisions based on bad assumptions. Some of the required estimations are the positive impact of a refactoring in future changes, the cost of adding new features and refactoring, the value for the customers of the features, and the team capacity. The goal of the present study was not to analyze estimation methods. We used methods that were previously used by other authors in technical debt management studies. The model is independent of the estimation methods used so that other techniques, even future techniques, could be used. Also, to establish a means with which to validate the estimations performed in the context of specific projects would help the process of technical debt management. This implies future studies focused on obtaining empirical evidences, validations of metrics, and estimation methods. We used conservative approaches in the estimation of principal and interest (see Section 7.5.2 and 7.5.3) to reduce the risk of identification of unnecessary refactorings, that is, false positives. We preferred to let a technical debt item in the product to recommend unnecessary changes. Refactoring incorporates its own risks in the possibility of adding bugs. Additionally, the effort spent in refactoring is subtracted from the available effort to add new functionality. Nevertheless, many times, more detailed data are subjective information. For example, effort is many times collected manually by developers. This means that these data could be imprecise. In this study, we have made some estimations that could also be imprecise, but they are based on objective data obtained from the source code.

When the team and managers have more experience, more accurate estimations can be performed. All these approximations have been based on ones found in the literature when the authors faced similar situations. Another assumption is that no business model is used. Whereas this may seem strange when a new product is developed, the finding reported by Hacklin and Wallnöfer [47] should be considered: the application of business models can be useful, but participants should be trained in advance. Otherwise, estimations may lead to wrong conclusions. Therefore, the application of the model to reason about technical debt management and time-to-market can be used as a first approximation. More accurate approximations could be obtained when more experience is gained from working with the product.

In the present case study, the model was adapted to be used in a context of few sources of information; if more information was available, the model could be used as defined in Section 7.3.

7.6.3 The nature of technical debt management

This work contributes by providing more input on the nature of technical debt management. The investment in infrastructure is represented by the parameter K in Equation 7.6, where K was the constant of proportionality for the speed of performance improvement. As mentioned, when this equation was described, K is proportional to the level of capital investment in the development technology. It was interpreted that the effort spent in fixing technical debt was invested in improving infrastructure, which makes sense because if legacy code is improved, productivity will also improve. From a business perspective, software refactoring can be seen as any other capital investment in the development technology of the software product process, for example, investments in tools or workstations that increase the team's development capacity.

7.6.4 The view of legacy code as product and as infrastructure

A final issue to highlight is that the source code not only be part of the current product software, but also a part of the infrastructure for the next releases. Actually, the investment in a current product to reduce technical debt can be assimilated to invest in the infrastructure to make the next version of this product. This interpretation of legacy code both as infrastructure and as a product version can be controversial but is valid from an economic perspective. This should be further investigated.

7.6.5 The role of technical debt management approaching internal and external quality

Technical debt management worked as a link between internal quality and external quality. Whereas internal and external quality were defined previously, no link between them was provided. Therefore, it was difficult to estimate how an improvement in internal quality could impact external quality, and as a result, it was difficult to justify how much should be invested to improve internal quality. This work provides a baseline with which to change this issue.

7.6.6 Costs and investments in software development

Although traditionally software development has been bound to an estimation of the cost associated with product development, this work provides a baseline for different costs, depending on how much technical debt is removed. Therefore, there could be different costs depending on this issue. If costs and investments (and the expected return on investment) are discussed together, it may help to better justify the job to be taken on and how much of an investment may make sense in reducing technical debt.

7.6.7 Extending the model

Cohen et al.'s model has been applied in two stages: technical debt management and development. However, Cohen et al.'s model could be applied in more stages. For the sake of the case study, it was not required, but when applying this model to the industry, it may help to have additional stages, such as testing.

7.6.8 The need for the concept of technical debt points

It was required to define and follow some conventions to interpret technical debt in the context of the releases. When the presented model is applied in industry, these conventions should be standardized within a company, and beyond. This is similar to what happened to function points: there may exist several conventions, for instance for function points we have several such as [53] and [52], but it is important to know which one is being used. Therefore, we realized that it is necessary to measure the "technical debt points" and identify the adopted convention.

7.6.9 Size of releases

It seems that short release planning could hinder the removal of technical debt. This does not mean that it is better to have large releases. A long-term perspective is very important in technical debt management, but short releases can lead to losing the long-term perspective; but if it is considered, for example, by analyzing several short releases in the roadmap of the product, this can be avoided. As shown in Section 7.5.5, few items are responsible for the most changes in the analyzed project. This indicates that focusing the effort in small refactorings could keep technical debt under control. This can provide insight into how to integrate technical debt management into software development processes.

7.6.10 Combination of anti-patterns

In Section 7.5.1, it is shown that combinations of some anti-patterns pointed to items with more technical debt than the individual anti-patterns. This is an interesting finding for technical debt identification. Nevertheless, there is no clear indication of what technical debt indicator to use; therefore, research should be performed on this topic. It is necessary to identify a set of technical debt indicators (not limited to anti-patterns) that allows for analysis of all possible causes of technical debt.

7.7 Limitations

Case studies allow us to evaluate a phenomenon, a model, or a process in a real setting. This is important in software engineering, in which a multitude of external factors may affect the validation results and where other techniques, such as formal experiments, are not considered to be conducted under controlled settings, even though formal experiments permit replication and generalization. In this sense, the major limitation concerns external validity because only one software product was considered. However, the objective was not to study the software product, but rather to explore a new way to model the relation between technical debt management and time-to-market under economic constraints to support decision making about how much technical debt must be removed and its economic consequences.

As part of the exploration, we reduced the problem to the analysis of technical debt in *Java* files in a well-known open source project. If other types of files were causing problems in that project, they were not considered. As a result, the outcomes of this study cannot be used to make decisions on the used project, but they can be used as a first input. Again, the objective of the exploratory study was not to study a software product, but rather to explore a new model. A complete analysis of all the files together with the knowledge of the developers would be necessary to make formed decisions. Furthermore, the tools used for identifying problems can produce false negatives or false positives. Therefore, the results can be affected by the effectiveness of the tools used.

In addition, it was found that files with technical debt indicators have a bigger than expected size of change than files without them. But this could be produced by any other file characteristic, for instance, files that implement a functionality that changes periodically. These risks could be reduced by contrasting the identified problems with the development team.

Additionally, as discussed in Section 7.6.2, to develop the model and put it in practice, several estimation and simplifications were done, specifically cost estimations, following approximations already used in the literature. This simplification might lead to the wrong results when using the model. Though, it should be noted that in software engineering, cost estimations are typically imprecise [63]; therefore, because of the high degree of uncertainty that exists in software cost estimations, it seems reasonable to use simplifications in the model, even if they might reduce the accuracy of the provided results.

Because we did not have market information, we had to simplify the modeling framework used. This often is the case when a team starts to work with legacy code without further background, something occurring frequently in software engineering projects. So we could not establish a fixed time frame, including the exploitation phase of the product. For more complete analyses, this information would be required.

Finally, some of the elements for technical debt management were not used in this study. We did not analyze "When to implement decisions", "Technical debt evolution", "Technical debt visualization, and "Expert opinion". These elements are more focused on actively managing technical debt over the project's development. As our goal was to analyze the integration of the time-to-market modeling framework in a real large project, we analyzed just a snapshot of the project as the first step.

7.8 Related Work

There are several studies about decision making in technical debt management. For example, Brown et al. [18] provided an approach to make decisions about when to improve the software architecture in the context of iterative release planning. They described a trade-off involving architectural investment versus the delivery of end-user valued capabilities. However, they do not consider time-to-market in the trade-off. Another method is provided by Kazman et al. [59] to locate architectural sources of technical debt, quantify them, and quantify the ROI of removing these debts. Nevertheless, they do not consider time-to-market as a restriction to calculate the impact of removing technical debt.

As we showed in [39], only some studies have considered time-to-market in technical debt management. Guo et al. used time-to-market as a constraint, but they did not use it actively in their model to make decisions [46]. That is, it was considered as a fixed constraint while we used it as a parameter that can be changed to achieve the different goals of the project. In other studies, for example, Nord et al. [78] and Ramasubbu and Kemerer [90], time is used to model a time frame in which it is possible to analyze the cost and benefit of technical debt. The time frame can be established based on the retirement time of the software. We used the same approach but also used time-to-market in making decisions about how much debt is beneficial to pay off.

We used Schmid's technical debt model [97][98] as a starting point; by using its definition of internal quality, external quality, evolution, and refactoring. Nevertheless, Schmid's model does not consider time-to-market.

7.9 Conclusions and future work

This chapter presents a model for technical debt management that considers time-to-market. The methodology used, with the goal of finding constraints and limits that should be considered when using the model, was an exploratory case study. The model uses a technical debt management framework as a guide, which was useful in adapting the time-to-market modeling framework, which came from a different domain, to technical debt management.

This model is a step forward in making decisions in software engineering, especially in the case of trade-offs of internal and external quality of software when considering an economic point of view. It was possible to deploy the model for the technical debt management in a large project. The case study's execution demonstrates that it is possible to use the selected time-to-market modeling framework for technical debt management and that it is necessary a holistic perspective in technical debt management that includes the business perspective, that is, economic constraints and business goals. This holistic perspective was clear when we had to make simplifications in the model because we did not have the business strategy information. So we could not establish a fixed time frame for the analysis, including the exploitation phase of the product. For more complete analyses, this information would be required. The model can be used in any other projects to manage technical debt. For those projects in which a team has to work with legacy code without further background and little or none productivity and market information, this paper provides the approximations for applying the model in a systematic fashion. This lack of information occurs frequently in software engineering projects.

The findings of this study (see Section 7.6) open new research opportunities that can be extended in the future. Further research is necessary in several topics: to analyze the relationships between internal and external quality of software; to study the effect of combinations of technical debt indicators in the same files; to study the paradox of source code not only being part of the current product software, but also being part of the infrastructure for the next releases; to define the methods for measuring the "technical debt points"; and to integrate technical debt management into software development processes.

As for future work, it is planned to apply the model to more projects with the same available information, that is, with only access to the source code repository and the track tool; this will be done to obtain more knowledge on the model's variables and parameters. With the same model and more information, a more complex scenario analysis can be performed. Therefore, we also intend to study products for which we have more information about, including team size, labor cost, and market data, so that we can work with more complex scenarios. In this case, methods such as Monte Carlo could help to simulate these complex scenarios with multiple variables. We are also working to have new case studies that include the participation of development teams. This will also require tuning the model to represent the inputs from the development team.

Part VI

Conclusion and Further Work

Chapter 8

Conclusions and Future Work

This chapter presents and analyses the main contributions of this thesis and it also presents future research to extend results that have already been obtained. Conclusions are framed respecting to the objectives proposed at the beginning of the research and formulated as research questions.

8.1 Research contributions

This thesis answers the RQ 1: Could one have a model that considers customer value together with the short and long-term impact of decisions to help support the decision-making process in software evolution? What elements would this model be made of? by providing a model oriented to the value for the customer that helps make decisions in a context of continuous evolution of software taking into account internal characteristics of software and the short and long-term impact of such decisions. This model has been created following a theoretical framework that identifies the elements that are necessary to consider in the model definition.

The specific contributions are described in the following subsections.

8.1.1 Identification of how software internal quality increases the customer value

The value for the customer that is added by internal quality of software consists of the possible future value that could be added over the evolution of software. Good levels of internal quality will imply that external quality might be added easily in future evolution. Therefore, value is affected by the probability of future changes and the time frame in which those changes can happen.

Technical debt concept, through principal, interest, and interest probability, helps reason about the value of changes in the internal quality of software considering the probability of change and the time frame. Therefore, technical debt management is a key factor to make decisions in changes in software with a value-based perspective.

Delaying decisions may have value in contexts where there is much uncertainty. Therefore, not only principal, interest, and interest probability, but also other factors, as the better moment to implement the decisions, have to be considered when managing technical debt.

8.1.2 Identification and definition of the elements that are required to create models that help make decisions in software evolution

The elements that are required to effectively manage technical debt have been identified and defined. These elements define a framework and this framework could be used to produce specific decision-making models and methods or to assess existing ones.

In addition to the framework, an important finding of this chapter is that any decision about software evolution implies a trade-off between software release characteristics and technical debt removal (increasing internal quality). Additionally, the technical debt impact might be effective in allowing communication between the different stakeholders in a project to reason about this trade-off.

Another important conclusion is that technical debt is *context dependent.* This means that the context, which is difficult to define, must be part of the estimation model, includes issues such as the history of the product development, prospects, or time to market.

8.1.3 Identification of tools and strategies that support the elements identified in Contribution 2 and the lacks in this support

The available techniques for technical debt management identified in the current literature have been analyzed. This analysis shows that further studies are necessary to fully support technical debt management and also essential elements not currently covered, such as time-to-market. Other elements as *E4 interest probability*, *E7 expert opinion*, *E10 When to implement decisions*, *E11 Technical debt evolution*, and *E12 Technical debt visualization* also require more support, especially from the business organizational management point of view.

Different strategies are focused on different elements. There are not tools or strategies that support all the elements. Therefore, it is necessary to go further in the integration of tools and strategies to manage effectively technical debt. In this regard, in this thesis, a tool named TEDMA has been implemented. TEDMA provides the required support to use the technical debt management elements defined in Chapter 4 in real projects. In fact, TEDMA creates a platform to experiment with technical debt tools and strategies. The analysis of the support of tools and strategies for the technical debt management elements indicated the necessity of integrating several different techniques. TEDMA responds to that necessity by providing a tool that is designed to integrate third party tools. In fact, TEDMA facilitates the usage of the technical debt management elements in real projects.

8.1.4 Definition of a model for making decisions on software evolution using the elements identified in Contribution 2 and that integrates tools and strategies identified in Contribution 3

This contribution consists of the definition of a model oriented to the value for the customer that helps make decisions in a context of continuous evolution of software considering the internal quality of software and the short and long-term impact of these decisions. The model uses the technical debt management framework defined in

Contribution 2 as a guide, which was useful in adapting the time-to-market modeling framework, which came from a different domain, to technical debt management.

This model is a step forward in making decisions in software engineering, especially in the case of trade-offs of internal and external quality of software when considering an economic point of view. It was possible to use the model for the technical debt management in a case study using a large project. The case study's execution demonstrates that it is possible to use the selected time-to-market modeling framework for technical debt management and that it is necessary a holistic perspective in technical debt management that includes the business perspective, that is, economic constraints and business goals. This holistic perspective was clear when we had to make simplifications in the model because we did not have the business strategy information.

8.2 Future Work

Technical debt helps reason about the value of changes in the internal quality of software. This thesis provides a theoretical framework that helps to manage technical debt. Following this framework, this thesis presents model for technical debt management considering one of the elements that did not have enough support by the available techniques and tools for technical debt management. But models for technical debt management should have the capacity to deal with all the elements. Therefore, new models, or an extension of the model presented in Chapter 7, have to be defined to achieve this goal.

It is planned to use the model presented in Chapter 7 in more case studies to find out more empirical evidences about the advantages and the limits of using the model for technical debt management. For example, it is planned to apply the model presented in this thesis to more projects with the same available information, that is, with only access to the source code repository and the track tool; this will be done to obtain more knowledge on the model's variables and parameters. With the same model and more information, a more complex scenario analysis can be performed. Therefore, we also intend to study products for which we have more information about, including team size, labor cost, and market data, so that we can work with more complex scenarios. In this case, methods such as Monte Carlo could help to simulate these complex scenarios with multiple variables. We are also working to have new case studies that include the participation of development teams. This will also require tuning the model to represent the inputs from the development team.

To do that, TEDMA, the tool presented in Chapter 6 will be extended to incorporate new strategies and tools for technical debt management. Currently, TEDMA is useful for experimenting in technical debt management using empirical data. The expected evolution of TEDMA will make it useful for software development industry. Using TEDMA, organizations will be able to manage the technical debt of their projects by selecting the tools and indicators that are the most important for them.

The findings of using the model presented in Chapter 7 in this thesis open new research opportunities that can be extended in the future. Further research is necessary in several topics: to extend the analysis of the relationships between internal and external quality of software; to study the effect of combinations of technical debt indicators in the same files; to study the paradox of source code not only being part of the current product software, but also being part of the infrastructure for the next releases; to define the methods for measuring the "technical debt points"; and to integrate technical debt management into software development processes.

From software industry perspective it is possible to extend this work to three interesting domains:

- Software startups. This is an interesting domain where this thesis can be extended. In the software startup context, it may be vital to be the first to market in order to obtain customers. Since software startups also lack resources, quality assurance is often largely absent [2]. However, long-term problems will only be relevant if the product obtains customers in the short term [33]. This short-term vision may produce software code with low internal quality and that is difficult to change, compelling the company to invest all of its efforts into keeping the system running, rather than increasing its value by adding new capabilities [33]. Scaling-up the system may become an obstacle, which will prevent the company from gaining new customers. Finding a viable trade-off between time-to-market demands and evolution needs is thus vital for software startups. The approach presented in this thesis can be used to perform this trade-off. The first step in this direction was performed by collaborating in one article to define a software startup research agenda [116].

- Internet of Things. By extending the research to Internet of Things domain it will be possible to identify new challenges about how technical debt management can be integrated into a domain where concurrent hardware and software engineering is performed.

- DevOps. Devops is about aligning the incentives of everybody involved in delivering software, with a particular emphasis on developers, testers, and operations personnel [50]. Studying how technical debt management might be integrated into this new development paradigm will allow going further in technical debt management and how it can be integrated into contexts where software is continuously delivered.

References

[1] "Iso/iec 25000 - software engineering - software product quality requirements and evaluation (square) - guide to square," tech. rep., ISO/IEC, 2005. 4

[2] "Iso/iec/ieee 42010:2011 systems and software engineering – architecture description," tech. rep., 1 2011. 17, 24

[3] "Mendeley," June 2012. `http://www.mendeley.com/`. 28, 29

[4] "Rapidminer," June 2012. `http://www.rapidminer.com/`. 27

[5] Alves, N. S. R., Ribeiro, L. F., Caires, V., Mendes, T. S., and Spínola, R. O., "Towards an ontology of terms on technical debt," in *International Workshop on Managing Technical Debt (MTD)*, 2014. 78

[6] Alves, N. S., Mendes, T. S., de Mendona, M. G., Spnola, R. O., Shull, F., and Seaman, C., "Identification and management of technical debt: A systematic mapping study," *Information and Software Technology*, vol. 70, pp. 100 – 121, 2016. 78, 94, 95, 107, 119, 145

[7] Ameller, D., Farré, C., Franch, X., and Rufian, G., "A survey on software release planning models," in *Proceedings of the Internantional Conference on Product-Focused Software Process Improvement, PROFES*, pp. 48–65, 2016. 5, 6, 137, 163

[8] Amin, S. M. and Wollenberg, B. F., "Toward a smart grid: power delivery for the 21st century," *IEEE Power and Energy Magazine*, vol. 3, pp. 34–41, Sept 2005. 55

[9] Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., and Avgeriou, P., "The financial aspect of managing technical debt," *Inf. Softw. Technol.*, 2015. 78, 145

[10] Babar, M., Zhu, L., and Jeffery, R., "A framework for classifying and comparing software architecture evaluation methods," in *Software Engineering Conference, 2004. Proceedings. 2004 Australian*, pp. 309 – 318, 2004. xv, 39, 41

[11] Bass, L., Clements, Paul, and Kazman, K., *Software Architecture in Practice, Third Edition.* Addison-Wesley Professional, 2012. 18, 24

[12] Benestad, H. C., Anda, B., and Arisholm, E., "Understanding software maintenance and evolution by analyzing individual changes: a literature review," *Journal of Software Maintenance and Evolution: Research and Practice*, vol. 21, no. 6, pp. 349–378, 2009. 6

[13] Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., and Grnbacher, P., *Value-Based Software Engineering.* Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2005. 6, 25

[14] Boehm, B., "Value-based software engineering: Overview and agenda," in *Value-Based Software Engineering* (Biffl, S., Aurum, A., Boehm, B., Erdogmus, H., and Grnbacher, P., eds.), pp. 3–14, Springer Berlin Heidelberg, 2006. 26

[15] Bosch, J., ed., *Continuous Software Engineering.* Springer, 2014. 4, 137

[16] Bourque, P. and Fairley, R. E., *Guide to the Software Engineering Body of Knowledge - SWEBOK v3.0.* IEEE CS, 2014 version ed., 2014. 89

[17] Breivold, H. P., Crnkovic, I., and Larsson, M., "A systematic review of software architecture evolution research," *Information and Software Technology*, vol. 54, no. 1, pp. 16 – 40, 2012. 6

[18] Brown, N., Nord, R. L., Ozkaya, I., and Pais, M., "Analysis and management of architectural dependencies in iterative release planning," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, pp. 103–112, June 2011. 168

[19] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., and Zazworka, N., "Managing technical debt in software-reliant systems," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10, (New York, NY, USA), pp. 47–52, ACM, 2010. 8, 137

[20] Buschmann, F., "To pay or not to pay technical debt," *IEEE Software*, vol. 28, pp. 29–31, Nov 2011. 8

[21] Cai, Y., Kazman, R., Silva, C. V., Xiao, L., and Chen, H.-M., "Chapter 6 - a decision-support system approach to economics-driven modularity evaluation," in *Economics-Driven Software Architecture*, 2014. 8

[22] Carriere, J., Kazman, R., and Ozkaya, I., "A cost-benefit framework for making architectural decisions in a business context," in *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering - Volume 2*, ICSE '10, (New York, NY, USA), pp. 149–157, ACM, 2010. 110

[23] Clements, P. and Northrop, L., *Software Product Lines: Practices and Patterns.* Addison-Wesley Professional, 2001. 89

[24] Cohen, M. A., Eliasberg, J., and Ho, T.-H., "New product development: The performance and time-to-market tradeoff," *Management Science*, vol. 42, no. 2, pp. 173–186, 1996. xiii, 137, 138, 140, 141, 145, 162

[25] Cohen, M. A., Eliasberg, J., and Ho, T.-H., "An analysis of several new product performance metrics," *Manufacturing & Service Operations Management*, vol. 2, no. 4, pp. 337–349, 2000. 138, 140, 145, 162

[26] Creswell, J., *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches.* SAGE Publications, 2014. 15

[27] Cruzes, D. S. and Dybå, T., "Research synthesis in software engineering: A tertiary study," *Inf. Softw. Technol.*, vol. 53, pp. 440–455, May 2011. 33

[28] Cunningham, W., "The wycash portfolio management system," *SIGPLAN OOPS Mess.*, vol. 4, pp. 29–30, Dec. 1992. 7

[29] den Ouden, E., *Innovation Design - Creating Value for People, Organizations and Society.* Springer London Dordrecht Heidelberg New York, 2012. 25

[30] Dobrica, L. and Niemela, E., "A survey on software architecture analysis methods," *Software Engineering, IEEE Transactions on*, vol. 28, pp. 638 – 653, jul 2002. 39

[31] Dybå, T. and Dingsøyr, T., "Empirical studies of agile software development: A systematic review," *Inf. Softw. Technol.*, vol. 50, pp. 833–859, August 2008. 197

[32] Emery, D., Hilliard, RichardF., I., and Rice, T., "Experiences applying a practical architectural method," in *Reliable Software Technologies Ada-Europe '96*

(Strohmeier, A., ed.), vol. 1088 of *Lecture Notes in Computer Science*, pp. 471–484, Springer Berlin Heidelberg, 1996. 24

[33] Falessi, D., Cantone, G., and Kruchten, P., "Do architecture design methods meet architects' needs?," in *Software Architecture, 2007. WICSA '07. The Working IEEE/IFIP Conference on*, p. 5, jan. 2007. 24

[34] Fernández, C., López, D., Yagüe, A., and Garbajosa, J., "Towards estimating the value of an idea," in *Proceedings of the 12th International Conference on Product Focused Software Development and Process Improvement*, Profes '11, (New York, NY, USA), pp. 62–67, ACM, 2011. 11, 14

[35] Fernández-Sánchez, C., Díaz, J., Pérez, J., and Garbajosa, J., "Guiding flexibility investment in agile architecting," in *2014 47th Hawaii International Conference on System Sciences*, pp. 4807–4816, Jan 2014. 11, 14, 49

[36] Fernández-Sánchez, C., Garbajosa, J., Vidal, C., and Yagüe, A., "An analysis of techniques and methods for technical debt management: A reflection from the architecture perspective," in *2015 IEEE/ACM 2nd International Workshop on Software Architecture and Metrics*, pp. 22–28, May 2015. 12, 14, 93, 95, 105

[37] Fernández-Sánchez, C., Garbajosa, J., and Yagüe, A., "A framework to aid in decision making for technical debt management," in *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, pp. 69–76, Oct 2015. 12, 14, 77, 130

[38] Fernández-Sánchez, C., Díaz, J., Garbajosa, J., and Pérez, J., "A cost-benefit analysis model for technical debt management considering uncertainty and time," in *Work in Progress Track at the Thirty Ninth Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, 2013. 11, 14, 49

[39] Fernández-Sánchez, C., Garbajosa, J., Yagüe, A., and Perez, J., "Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study," *Journal of Systems and Software*, vol. 124, pp. 22 – 38, 2017. xiii, 6, 13, 14, 77, 105, 130, 143, 144, 145, 148, 149, 163, 168

[40] FindBugs, "Findbugs web project." 123, 127

[41] Fitzgerald, B. and Stol, K.-J., "Continuous software engineering: A roadmap and agenda," vol. 123, pp. 176 – 189, 2017. 4, 5, 6, 7, 137

[42] Garlan, D., *Formal Modeling and Analysis of Software Architecture: Components, Connectors, and Events*, pp. 1–24. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. 17

[43] Git, "Git web page." 123

[44] Glass, R. L., "Frequently forgotten fundamental facts about software engineering," *IEEE Software*, vol. 18, pp. 112–111, May 2001. 5

[45] Gorton, I., *Essential Software Architecture, second edition.* Springer-Verlag, 2011. 24

[46] Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., da Silva, F., Santos, A., and Siebra, C., "Tracking technical debt: An exploratory case study," in *ICSM*, 2011. 168

[47] Hacklin, F. and Wallnöfer, M., "The business model in the practice of strategic decision making: insights from a case study," *Management Decision*, vol. 50, no. 2, pp. 166–188, 2012. 164

[48] Hofmeister, C., Kruchten, P., Nord, R. L., Obbink, H., Ran, A., and America, P., "Generalizing a model of software architecture design from five industrial approaches," in *Proceedings of the 5th Working IEEE/IFIP Conference on Software Architecture*, WICSA '05, (Washington, DC, USA), pp. 77–88, IEEE Computer Society, 2005. 25

[49] Hull, J. C., *Options, Futures, and Other Derivatives.* Prentice Hall, seventh ed., 2009. 42

[50] Humble, J. and Molesky, J., "Why eenterprise must adopt devops to enable continuous delivery," vol. 24, no. 8, pp. 6 – 12, 2011. 178

[51] Ionita, M., America, P., Obbink, H., and Hammer, D., "Quantitative architecture usability assessment with scenarios," in *CLOSING THE GAPS: Software Engineering and Human-Computer Interaction workshop, INTERACT 2003*, 2003. 41, 43

[52] ISO, "Information technology — systems and software engineering — fisma 1.1 functional size measurement method," ISO 19761:2011, International Organization for Standardization, Geneva, Switzerland, 2011. 166

[53] ISO, "Software engineering. a functional size measurement method," ISO 19761:2011, International Organization for Standardization, Geneva, Switzerland, 2011. 166

[54] ITEA, "08022 nemo coded." 55

[55] ITEA, "09030 imponet." 55

[56] Ivarsson, M. and Gorschek, T., "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Software Engineering*, vol. 16, no. 3, pp. 365–395, 2011. 16, 106, 107

[57] Jan, N. and Ibrar, M., "Systematic mapping of value-based software engineering - a systematic review of value-based requirements engineering," Master's thesis, School of Computing at Blekinge Institute of Technology, 2010. 27

[58] JGit, "Jgit web project." 126, 128

[59] Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., Fedak, V., and Shapochka, A., "A case study in locating the architectural roots of technical debt," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 179–188, May 2015. 130, 150, 154, 155, 168

[60] Kazman, R., Klein, M., Barbacci, M., Longstaff, T., Lipson, H., and Carriere, J., "The architecture tradeoff analysis method," in *Engineering of Complex Computer Systems, 1998. ICECCS '98. Proceedings. Fourth IEEE International Conference on*, pp. 68 –78, aug 1998. 41, 43

[61] Kazman, R., Bass, L., Klein, M., Lattanze, T., and Northrop, L., "A basis for analyzing software architecture analysis methods," *Software Quality Journal*, vol. 13, pp. 329–355, 2005. 10.1007/s11219-005-4250-1. xv, 39, 43

[62] Khomh, F., Penta, M. D., and Gueheneuc, Y. G., "An exploratory study of the impact of code smells on software change-proneness," in *2009 16th Working Conference on Reverse Engineering*, pp. 75–84, Oct 2009. 109

[63] Kitchenham, B. and Charters, S., "Guidelines for performing systematic literature reviews in software engineering," Tech. Rep. EBSE 2007-001, Keele University and Durham University Joint Report, 2007. 15, 16, 17, 24, 26, 27, 167

[64] Kruchten, P., Nord, R. L., and Ozkaya, I., “Technical debt: From metaphor to theory and practice,” *IEEE Software*, vol. 29, pp. 18–21, Nov 2012. 50, 55, 137, 154

[65] Leffingwell, D., *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Addison-Wesley Professional, 1st ed., 2011. 61

[66] Lehman, M. M. and Belady, L. A., eds., *Program Evolution: Processes of Software Change.* San Diego, CA, USA: Academic Press Professional, Inc., 1985. 5

[67] Lehman, M. M. and Ramil, J. F., “Software evolution and software evolution processes,” *Ann. Softw. Eng.*, vol. 14, pp. 275–309, Dec. 2002. 4, 137

[68] Li, W. and Shatnawi, R., “An empirical study of the bad smells and class error probability in the post-release object-oriented system evolution,” *J. Syst. Softw.*, vol. 80, pp. 1120–1128, July 2007. 109

[69] Li, Z., Avgeriou, P., and Liang, P., “A systematic mapping study on technical debt and its management,” *Journal of Systems and Software*, vol. 101, pp. 193 – 220, 2015. 78, 81, 94, 107, 130, 140, 145, 148

[70] MacCormack, A., Crandall, W., Henderson, P., and Toft, P., “Do you need a new product-development strategy?,” *Research-Technology Management*, vol. 55, no. 1, pp. 34–43, 2012. 138

[71] MacCormack, A., Rusnak, J., and Baldwin, C. Y., “Exploring the structure of complex software designs: An empirical study of open source and proprietary code,” *Manage. Sci.*, vol. 52, pp. 1015–1030, July 2006. 110

[72] Macia, I., Arcoverde, R., Garcia, A., Chavez, C., and von Staa, A., “On the relevance of code anomalies for identifying architecture degradation symptoms,” in *2012 16th European Conference on Software Maintenance and Reengineering*, pp. 277–286, March 2012. 109

[73] Marinescu, R., “Assessing technical debt by identifying design flaws in software systems,” *IBM Journal of Research and Development*, vol. 56, pp. 9:1–9:13, Sept. 2012. 149

[74] Martin, J., Yague, A., Gonzalez, E., and Garbajosa, J., “Making software factory truly global: the smart software factory project,” *Software Factory Magazine*, p. 19, 2010. 55

[75] MILES, L., *Techniques of value analysis and engineering.* New York [etc.] :: McGraw-Hill,, 1972. 25

[76] NEO4J, "Neo4j web project." 126, 128

[77] NISTOR, A., CHANG, P.-C., RADOI, C., and LU, S., "Caramel: Detecting and fixing performance problems that have non-intrusive fixes," in *Proceedings of the 37th International Conference on Software Engineering - Volume 1*, ICSE '15, (Piscataway, NJ, USA), pp. 902–912, IEEE Press, 2015. 145

[78] NORD, R., OZKAYA, I., KRUCHTEN, P., and GONZALEZ-ROJAS, M., "In search of a metric for managing architectural technical debt," in *WICSA/ECSA*, 2012. 168

[79] OLBRICH, S., CRUZES, D. S., BASILI, V., and ZAZWORKA, N., "The evolution and impact of code smells: A case study of two open source systems," in *2009 3rd International Symposium on Empirical Software Engineering and Measurement*, pp. 390–400, Oct 2009. 109

[80] OLBRICH, S. M., CRUZES, D. S., and SJBERG, D. I. K., "Are all code smells harmful? a study of god classes and brain classes in the evolution of three open source systems," in *2010 IEEE International Conference on Software Maintenance*, pp. 1–10, Sept 2010. 109

[81] PATERNOSTER, N., GIARDINO, C., UNTERKALMSTEINER, M., GORSCHEK, T., and ABRAHAMSSON, P., "Software development in startup companies: A systematic mapping study," *Information and Software Technology*, vol. 56, no. 10, pp. 1200 – 1218, 2014. 81, 106

[82] PEREZ, J., DIAZ, J., COSTA-SORIA, C., and GARBAJOSA, J., "Plastic partial components: A solution to support variability in architectural components," in *2009 Joint Working IEEE/IFIP Conference on Software Architecture European Conference on Software Architecture*, pp. 221–230, Sept 2009. 67

[83] PETERS, L., "Technical debt: The ultimate antipattern - the biggest costs may be hidden, widespread, and long term," in *2014 Sixth International Workshop on Managing Technical Debt*, pp. 8–10, Sept 2014. 8, 85

[84] PETERSEN, K., FELDT, R., MUJTABA, S., and MATTSSON, M., "Systematic mapping studies in software engineering," in *EASE*, 2008. xi, 16, 18, 79, 81, 82, 99, 106

[85] PMD, "Pmd web project." xvi, 62, 67, 123, 127, 149, 151

[86] Popay, J., Roberts, H., Sowden, A., Petticrew, M., Arai, L., Rodgers, M., Britten, N., Roen, K., and Duffy, S., "Guidance on the conduct of narrative synthesis in systematic reviews," tech. rep., ESRC Methods Programme, 2006. xi, 15, 16, 32, 33, 34, 35

[87] Power, K., "Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options," in *2013 4th International Workshop on Managing Technical Debt (MTD)*, pp. 28–31, May 2013. 8

[88] Qu, Y., Guan, X., Zheng, Q., Liu, T., Wang, L., Hou, Y., and Yang, Z., "Exploring community structure of software call graph and its applications in class cohesion measurement," *Journal of Systems and Software*, vol. 108, pp. 193 – 210, 2015. 145

[89] R, "The r project for statistical computing." 123

[90] Ramasubbu, N. and Kemerer, C., "Towards a model for optimizing technical debt in software products," in *International Workshop on Managing Technical Debt (MTD)*, 2013. 168

[91] Renjin, "Renjin web project." 128

[92] Riaz, M., Mendes, E., and Tempero, E., "A systematic review of software maintainability prediction and metrics," in *Proceedings of the 2009 3rd International Symposium on Empirical Software Engineering and Measurement*, ESEM '09, (Washington, DC, USA), pp. 367–377, IEEE Computer Society, 2009. 6

[93] Rserve, "Rserve web project." 128

[94] Runeson, P. and Höst, M., "Guidelines for conducting and reporting case study research in software engineering," *Empirical Softw. Engg.*, vol. 14, pp. 131–164, Apr. 2009. 17, 19, 54, 136, 143

[95] Saliu, O. and Ruhe, G., "Supporting software release planning decisions for evolving systems," in *29th Annual IEEE/NASA Software Engineering Workshop*, pp. 14–26, April 2005. 5, 6

[96] Saliu, O. and Ruhe, G., "Software release planning for evolving systems," *Innovations in Systems and Software Engineering*, vol. 1, no. 2, pp. 189–204, 2005. 137

[97] Schmid, K., "On the limits of the technical debt metaphor some guidance on going beyond," in *MTD workshop*, 2013. 168

[98] Schmid, K., "A formal approach to technical debt decision making," in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '13, (New York, NY, USA), pp. 153–162, ACM, 2013. 4, 25, 137, 154, 168

[99] Schumacher, J., Zazworka, N., Shull, F., Seaman, C., and Shaw, M., "Building empirical support for automated code smell detection," in *Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, ESEM '10, (New York, NY, USA), pp. 8:1–8:10, ACM, 2010. 109

[100] Schwaber, K. and Beedle, M., *Agile Software Development with Scrum.* Prentice Hall, 2002. 55

[101] Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., and Vetro, A., "Using technical debt data in decision making: Potential decision approaches," in *MTD Workshop*, 2012. 149

[102] Shull, F., Falessi, D., Seaman, C., Diep, M., and Layman, L., "Technical debt: Showing the way for better transfer of empirical results," in *Perspectives on the Future of Software Engineering* (Münch, J. and Schmid, K., eds.), pp. 179–190, Springer Berlin Heidelberg, 2013. 126

[103] Siebra, C., Cavalcanti, A., Silva, F., Santos, A., and Gouveia, T., "Applying metrics to identify and monitor technical debt items during software evolution," in *ISSREW, 2014*, pp. 92–95, Nov 2014. 149

[104] Skytap, "2015 software development survey," tech. rep., 2015. 137

[105] Smith, J. E. and Nau, R. F., "Valuing risky projects: Option pricing theory and decision analysis," *Manage. Sci.*, vol. 41, pp. 795–816, May 1995. 52

[106] Society, I. C., Bourque, P., and Fairley, R. E., eds., *Guide to the Software Engineering Body of Knowledge (SWEBOK(R)): Version 3.0.* Los Alamitos, CA, USA: IEEE Computer Society Press, 3rd ed., 2014. 152

[107] SONARQUBE, “Sonarqube web project.” 62, 67, 130

[108] STRAUSS, A., CORBIN, J., and OTHERS, *Basics of qualitative research*, vol. 15. Newbury Park, CA: Sage, 1990. 16, 18, 81

[109] SUOMALAINEN, T., SALO, O., ABRAHAMSSON, P., and SIMIL, J., “Software product roadmapping in a volatile business environment,” *Journal of Systems and Software*, vol. 84, no. 6, pp. 958 – 975, 2011. 94

[110] SVAHNBERG, M., GORSCHEK, T., FELDT, R., TORKAR, R., SALEEM, S. B., and SHAFIQUE, M. U., “A systematic review on strategic release planning models,” *Information and Software Technology*, vol. 52, no. 3, pp. 237 – 248, 2010. 5, 6, 137, 140

[111] TAIBI, D., LENARDUZZI, V., AHMAD, M. O., LIUKKUNEN, K., LUNESU, I., MATTA, M., FAGERHOLM, F., MUNCH, J., PIETINEN, S., TUKIAINEN, M., FERNANDEZ-SANCHEZ, C., GARBAJOSA, J., and SYSTA, K., “Free innovation environments: Lessons learned from the software factory initiatives,” in *ICSEA 2015, The Tenth International Conference on Software Engineering Advances*, 2015. 55

[112] TOCKEY, S., *Return On Software: Maximizing The Return On Your Software Investment.* Addison-Wesley Professional, 2005. 52

[113] TOCKEY, S., “Chapter 3 - aspects of software valuation,” in *Economics-Driven Software Architecture* (MISTRIK, I., BAHSOON, R., KAZMAN, R., and ZHANG, Y., eds.), pp. 37 – 58, Boston: Morgan Kaufmann, 2014. 8, 84

[114] TOM, E., AURUM, A., and VIDGEN, R., “An exploration of technical debt,” *Journal of Systems and Software*, vol. 86, no. 6, pp. 1498–1516, 2013. 8, 9, 50, 78, 83, 85, 87, 93, 95, 100, 137, 144, 145, 158

[115] TRICCO, A. C., TETZLAFF, J., SAMPSON, M., FERGUSSON, D., COGO, E., HORSLEY, T., and MOHER, D., “Few systematic reviews exist documenting the extent of bias: a systematic review,” *Journal of Clinical Epidemiology*, vol. 61, no. 5, pp. 422 – 434, 2008. 98

[116] UNTERKALMSTEINER, M., ABRAHAMSSON, P., WANG, X., NGUYEN-DUC, A., SHAH, S., BAJWA, S., BALTES, G., CONBOY, K., CULLINA, E., DENNEHY, D., EDISON, H., FERNANDEZ-SANCHEZ, C., GARBAJOSA, J., GORSCHEK, T., KLOTINS, E., HOKKANEN, L., KON, F., LUNESU, I., MARCHESI, M., MORGAN,

L., Oivo, M., Selig, C., Seppnen, P., Sweetman, R., Tyrvinen, P., Ungerer, C., and Yage, A., "Software startups-a research agenda," *E-Informatica Software Engineering Journal*, vol. 10, no. 1, pp. 89–123, 2016. cited By 1. 12, 14, 177

[117] Wohlin, C., "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14, (New York, NY, USA), pp. 38:1–38:10, ACM, 2014. 81

[118] Wong, S., Cai, Y., Kim, M., and Dalton, M., "Detecting software modularity violations," in *2011 33rd International Conference on Software Engineering (ICSE)*, pp. 411–420, May 2011. 109, 130

[119] Wong, S., Cai, Y., Valetto, G., Simeonov, G., and Sethi, K., "Design rule hierarchies and parallelism in software development tasks," in *2009 IEEE/ACM International Conference on Automated Software Engineering*, pp. 197–208, Nov 2009. 109

[120] Woodruff, R., "Customer value: The next source for competitive advantage," *Journal of the Academy of Marketing Science*, vol. 25, pp. 139–153, 1997. 10.1007/BF02894350. 25

[121] Xiao, L., Cai, Y., and Kazman, R., "Design rule spaces: A new form of architecture insight," in *Proceedings of the 36th International Conference on Software Engineering*, ICSE 2014, (New York, NY, USA), pp. 967–977, ACM, 2014. 109

[122] Xiao, L., Cai, Y., and Kazman, R., "Titan: A toolset that connects software architecture with quality analysis," in *Proceedings of the 22Nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, FSE 2014, (New York, NY, USA), pp. 763–766, ACM, 2014. 109

[123] Yin, R. K., *Case Study Research.* SAGE Publications Inc., fourth edition ed., 2009. 17

[124] Zazworka, N., Seaman, C., and Shull, F., "Prioritizing design debt investment opportunities," in *Proceedings of the 2Nd Workshop on Managing Technical Debt*, MTD '11, (New York, NY, USA), pp. 39–42, ACM, 2011. 51, 151

[125] Zazworka, N., Shaw, M. A., Shull, F., and Seaman, C., "Investigating the impact of design debt on software quality," in *Proceedings of the 2Nd Workshop*

on Managing Technical Debt, MTD '11, (New York, NY, USA), pp. 17–23, ACM, 2011. 109

[126] Zazworka, N., Vetro', A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., and Shull, F., "Comparing four approaches for technical debt identification," *Software Quality Journal*, vol. 22, pp. 403–426, Sept. 2014. 123, 127, 130, 149, 150, 151, 154

[127] Zhang, H. and Ali Babar, M., "On searching relevant studies in software engineering," 2010. 27

Declaration

I herewith declare that I have produced this paper without the prohibited assistance of third parties and without making use of aids other than those specified; notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in identical or similar form to any other Spanish or foreign examination board.

The thesis work was conducted under the supervision of Dr. Juan Garbajosa at Universidad Politécnica de Madrid.

Madrid, May 17st, 2017

Appendices

Appendix A

Data Extraction Form

This is the data extraction form used in the systematic literature review described in Chapter 2.

1. General information: basic information about the study analyzed.
   (a) Id
   (b) Analysis Date
   (c) Research type
   (d) Type of article
   (e) Are results described? (No, author statements, qualitative, quantitave, both)
2. Architecture: how the studies use the concept of software architecture is recorded.
   (a) Type of architecture
   (b) Architecture activity where the study is focus on
   (c) Definition of architecture if it is given
3. Product: Similar to the previous one, but in this case centered on the software product.
   (a) Type of product
   (b) Definition of product/system if it is given
4. Environment: where the study was performed.
   (a) Business area

   (b) Methodology
   (c) Technology
   (d) Country
   (e) Type of organization

5. Stakeholders: stakeholders identified by the studies.
   (a) Stakeholders identified
   (b) All stakeholders are taken into account?

6. Value: what type of value is used, when a study uses the concept of value?
   (a) Identified value
   (b) Levels and Perspectives
   (c) How value is measured or estimated?
   (d) Definition of value if it is given

7. Concerns: which concerns the study deals with, if any.
   (a) Is the study centered on some specific concerns?
   (b) What concerns are treated?
   (c) Definition of used concerns if it given

Appendix B

Quality Questions

Quality questions, based on a systematic literature review performed by Tore Dybå and Torgeir Dingsøyr [31], were used for the systematic literature review presented in Chapter 2.

1. Is this a research paper?
   (a) Is the paper based on research (or is it merely a “lessons learned” report based on expert opinion?
2. Is there a clear statement of the aims of the research?
   (a) Is there a rationale for why the study was undertaken?
   (b) Is there a clear statement of the study’s primary outcome (i.e., time-to-market, cost, or product or process quality)?
3. Is there an adequate description of the context in which the research was carried out?
   (a) The industry in which products are used (e.g., banking, telecommunications, consumer goods, travel, etc.)
   (b) The nature of the software development organization (e.g. in-house department or independent software supplier)
   (c) The skills and experience of software staff (e.g, with a language, a method, a tool, an application domain)
   (d) The type of software products used (e.g., a design tool, a compiler)
   (e) The software processes used (e.g., a company standard process, the quality assurance procedures, the configuration management process)

4. Was the research design appropriate to address the aims of the research?
    (a) Has the researcher justified the research design (e.g., have they discussed how they decided which methods to use)?
    (b) Is the research design appropriate for the research goals?
5. Was the recruitment strategy appropriate to the aims of the research?
    (a) Has the researcher explained how the participants or cases were identified and selected?
    (b) Are the cases defined and described precisely?
    (c) Were the cases representative of a defined population?
    (d) Have the researchers explained why the participants or cases they selected were the most appropriate to provide access to the type of knowledge sought by the study?
    (e) Was the sample size sufficiently large?
6. Was there a control group with which to compare treatments?
    (a) How were the controls selected?
    (b) Were they representative of a defined population?
7. Were the data collected in a way that addressed the research issue?
    (a) Were all measures clearly defined (e.g., units and counting rules)?
    (b) Is it clear how data were collected (e.g., semi-structured interviews, focus groups etc.)?
    (c) Has the researcher justified the methods that were chosen?
    (d) Has the researcher made the methods explicit (e.g., is there an indication of how interviews were conducted; did they use an interview guide)?
    (e) Whether the form of the data is clear (e.g., tape recording, video material, notes etc.)
    (f) Whether quality control methods were used to ensure completeness and accuracy of data collection
8. Were the data analysis sufficiently rigorous?
    (a) Was there an in-depth description of the analysis process?

   (b) Has sufficient data been presented to support the findings?
   (c) To what extent has contradictory data been taken into account?
   (d) Were quality control methods used to verify the results?

9. Has the relationship between researcher and participants been considered adequately?
   (a) Did the researcher critically examine their own role, potential bias and influence during the formulation of the research questions, sample recruitment, data collection, analysis, and the selection of data for presentation?

10. Is there a clear statement of the findings?
   (a) Are the findings explicit (e.g., magnitude of effect)?
   (b) Has an adequate discussion of the evidence, both for and against the researcher's arguments, been demonstrated?
   (c) Has the researcher discussed the credibility of the findings (e.g., triangulation, respondent validation, more than one analyst, etc.)?
   (d) Are the limitations of the study discussed explicitly?
   (e) Are the findings discussed in relation to the original research questions?
   (f) Are the conclusions justified by the results?

11. Is the study of value for research or practice?
   (a) Does the researcher discuss the contribution the study makes to existing knowledge or understanding (e.g., Do they consider the findings in relation to current practice or relevant research-based literature)?
   (b) Does the research identify new areas in which research is necessary?
   (c) Does the researcher discuss whether or how the findings can be transferred to other populations and consider other ways in which the research can be used?

Appendix C

Studies Included in Literature Reviews

C.1 Selected Studies in SLR

[S1] Abowd, G., Pitkow, J., and Kazman, R., "Analyzing differences between Internet information system software architectures," in *Proceedings of ICC/SUPERCOMM '96 - International Conference on Communications*, vol. 1, pp. 203–207, IEEE, June 1996.

[S2] Al-Naeem, T., Gorton, I., Babar, M. A., Rabhi, F., and Benatallah, B., "A quality-driven systematic approach for architecting distributed software applications," in *Software Engineering, 2005. ICSE 2005. Proceedings. 27th International Conference on*, pp. 244–253, May 2005.

[S3] Alzaghoul, E. and Bahsoon, R., "CloudMTD: Using real options to manage technical debt in cloud-based service selection," in *2013 4th International Workshop on Managing Technical Debt (MTD)*, pp. 55–62, IEEE, May 2013.

[S4] Andrews, A., Mancebo, E., Runeson, P., and France, R., "A Framework for Design Tradeoffs," *Software Quality Journal*, vol. 13, no. 4, pp. 377–405, 2005.

[S5] Bahsoon, R., Emmerich, W., and Macke, J., "Using real options to select stable middleware-induced software architectures," *IEE Proceedings - Software*, vol. 152, no. 4, pp. 167–186, 2005.

[S6] Bahsoon, R. and Emmerich, W., "ArchOptions: A Real Options-Based Model for Predicting the Stability of Software Architecture," in *Proceedings of the Fifth Workshop on Economics-Driven Software Engineering Research, EDSER 5, held in conjunction with the 25 th International Conference on Software Engineering*, 2003.

[S7] Bahsoon, R. and Emmerich, W., "An economics-driven approach for valuing scalability in distributed architectures," in *7th IEEE/IFIP Working Conference on Software Architecture, WICSA 2008*, (Vancouver, BC, Canada), pp. 9–18, 2008.

[S8] Baldwin, C. Y. and Clark, K. B., "The architecture of participation: Does code architecture mitigate free riding in the open source development model?," *Management Science*, vol. 52, no. 7, pp. 1116–1127, 2006.

[S9] Berry, D., Hungate, C., and Temple, T., "Delivering expected value to users and stakeholders with User Engineering," *IBM Systems Journal*, vol. 42, no. 4, pp. 542–567, 2003.

[S10] Brown, N., Nord, R. L., Ozkaya, I., and Pais, M., "Analysis and Management of Architectural Dependencies in Iterative Release Planning," in *2011 Ninth Working IEEE/IFIP Conference on Software Architecture*, (Boulder, CO), pp. 103–112, IEEE, June 2011.

[S11] Cai, Y. and Sullivan, K., "A formal model for automated software modularity and evolvability analysis," *ACM Transactions on Software Engineering and Methodology*, vol. 21, pp. 1–29, Nov. 2012.

[S12] Carriere, J., Kazman, R., and Ozkaya, I., "A cost-benefit framework for making architectural decisions in a business context," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, vol. 2, (New York, New York, USA), p. 149, ACM Press, May 2010.

[S13] Cortellessa, V., Marinelli, F., and Potena, P., "An optimization framework for build-or-buy decisions in software architecture," *Computers &amp; Operations Research*, vol. 35, no. 10, pp. 3090–3106, 2008.

[S14] Diaz-Pace, J. A., Nicoletti, M., Schiaffino, S., Villavicencio, C., Sanchez, L. E., Andres Diaz-Pace, J., and Emiliano Sanchez, L., "A Stakeholder-Centric Optimization Strategy for Architectural Documentation," *Model and Data Engineering, Medi 2013*, vol. 8216, pp. Univ Calabria, Dipartimento Ingegneria Informatica, 2013.

[S15] Eklund, U. and Arts, T., "A classification of value for software architecture decisions," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6285 LNCS, pp. 368–375, Aug. 2010.

[S16] Engel, A. and Browning, T. R., "Designing systems for adaptability by means of architecture options," *Systems Engineering*, vol. 11, no. 2, pp. 125–146, 2008.

[S17] Falessi, D., Cantone, G., and Kruchten, P., "Value-Based Design Decision Rationale Documentation: Principles and Empirical Feasibility Study," in *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, (Vancouver, BC), pp. 189–198, IEEE, Feb. 2008.

[S18] Falessi, D., Capilla, R., and Cantone, G., "A value-based approach for documenting design decisions rationale: A replicated experiment," in *Proceedings - International Conference on Software Engineering*, (Leipzig), pp. 63–69, 2008.

[S19] Fernandez-Sanchez, C., Diaz, J., Perez, J., Garbajosa, J., Fernández-Sánchez, C., Díaz, J., and Pérez, J., "Guiding Flexibility Investment in Agile Architecting," in *2014 47th Hawaii International Conference on System Sciences*, HICSS '14, (Washington, DC, USA), pp. 4807–4816, IEEE, Jan. 2014.

[S20] Gonzalez-Huerta, J., Insfran, E., Abrahão, S., and Scanniello, G., "Validating a Model-Driven Software Architecture Evaluation and Improvement Method: A Family of Experiments," *Information and Software Technology*, no. 0, pp. –, 2014.

[S21] Gordijn, J., Akkermans, H., and Van Vliet, H., "Value based requirements creation for electronic commerce applications," in *Proceedings of the 33rd Annual Hawaii International Conference on System Sciences*, vol. vol.1, (Maui, USA), p. 10, IEEE Comput. Soc, 2000.

[S22] Gustavsson, H. k. and Axelsson, J., "Evaluating Flexibility in Embedded Automotive Product Lines Using Real Options," in *2008 12th International Software Product Line Conference*, (Limerick), pp. 235–242, IEEE, Sept. 2008.

[S23] Gustavsson, H. and Axelsson, J., "Improving the System Architecting Process through the Use of Lean Tools," *Picmet 2010: Technology Management For Global Economic Growth*, p. Natl Sci Technol & Innovation Policy Off (STI); Si, 2010.

[S24] Ionita, M., America, P., Hammer, D., Obbink, H., and Trienekens, J., "A scenario-driven approach for value, risk, and cost analysis in system architecting for innovation," in *Proceedings. Fourth Working IEEE/IFIP Conference on Software Architecture (WICSA 2004)*, pp. 277–280, IEEE Comput. Soc, 2004.

[S25] Ivanovic, A. and America, P., "Customer value in architecture decision making," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 6285 LNCS, (Copenhagen, Denmark), pp. 263–278, 2010.

[S26] Ivanovic, A., America, P., and Snijders, C., "Modeling customer-centric value of system architecture investments," *Software and Systems Modeling*, vol. 12, pp. 369–385, May 2013.

[S27] Kazman, R., Asundi, J., and Klein, M., "Quantifying the costs and benefits of architectural decisions," in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001*, (Toronto, Ont), pp. 297–306, IEEE Comput. Soc, May 2001.

[S28] Kazman, R., Barbacci, M., Klein, M., Carrière, S. J., and Woods, S. G., "Experience with performing architecture tradeoff analysis," in *Proceedings - International Conference on Software Engineering*, (Los Angeles, CA, USA), pp. 54–63, IEEE, Los Alamitos, CA, United States, 1999.

[S29] Kazman, R., In, H. P., and Chen, H.-M., "From requirements negotiation to software architecture decisions," *Information and Software Technology*, vol. 47, pp. 511–520, June 2005.

[S30] Kim, C.-K., Lee, D.-H., Ko, I.-Y., and Baik, J., "A Lightweight Value-based Software Architecture Evaluation," in *Eighth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing (SNPD 2007)*, vol. 2, (Qingdao), pp. 646–649, IEEE, July 2007.

[S31] Koziolek, A., "Research preview: Prioritizing quality requirements based on software architecture evaluation feedback," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7195 LNCS, pp. 52–58, 2012.

[S32] Koziolek, H., Domis, D., Goldschmidt, T., Vorst, P., and Weiss, R. J., "MORPHOSIS: A lightweight method facilitating sustainable software architectures," in *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012*, (Helsinki, Finland), pp. 253–257, 2012.

[S33] Langdon, C. S., "Designing information systems capabilities to create business value: A theoretical conceptualization of the role of flexibility and integration," *Journal of Database Management*, vol. 17, no. 3, pp. 1–18, 2006.

[S34] Lee, J., Kang, S., and Kim, C.-K., "Software architecture evaluation methods based on cost benefit analysis and quantitative decision making," *Empirical Software Engineering*, vol. 14, no. 4, pp. 453–475, 2009.

[S35] Lee, Y. and Choi, H.-J., "Experience of combining qualitative and quantitative analysis methods for evaluating software architecture," in *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pp. 152–157, IEEE, 2005.

[S36] Losavio, F., Chirinos, L., Matteo, A., Lévy, N., and Ramdane-Cherif, A., "ISO quality standards for measuring architectures," *Journal of Systems and Software*, vol. 72, no. 2, pp. 209–223, 2004.

[S37] Losavio, F., Chirinos, L., Lévy, N., and Ramdane-Cherif, A., "Quality characteristics for software architecture," *Journal of Object Technology*, vol. 2, no. 2, pp. 133–150, 2003.

[S38] Marinescu, R., "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development*, vol. 56, pp. 9:1–9:13, Sept. 2012.

[S39] Martínez-Fernández, S., Ayala, C., Franch, X. X., Marques, H. M. H., Ameller, D. D., and Martinez-Fernandez, S., "A framework for software reference architecture analysis and review," in *CIbSE 2013: 16th Ibero-American Conference on Software Engineering - Memorias del 10th Workshop*

Latinoamericano Ingenieria de Software Experimental, ESELAW 2013, (Montevideo, Uruguay), pp. 89–102, 2013.

[S40] Martinez-Fernandez, S., Ayala, C., Franch, X., and Marques, H. M., "REARM: a reuse-based economic model for software reference architectures," in *13th International Conference on Software Reuse, ICSR 2013, Pisa, June 18-20. Proceedings: LNCS 7925*, vol. 7925 LNCS, pp. 97–112, 2013.

[S41] Mavridis, A., Ampatzoglou, A., Stamelos, I., Sfetsos, P., and Deligiannis, I., "Selecting Refactorings: An Option Based Approach," in *2012 Eighth International Conference on the Quality of Information and Communications Technology*, pp. 272–277, IEEE, Sept. 2012.

[S42] McGregor, J. D., Monteith, J. Y., and Zhang, J., "Quantifying value in software product line design," in *15th International Software Product Line Conference, SPLC'11*, (Munich, Germany), pp. Siemens; Hitachi – Inspire the Next; Pure–Systems;, 2011.

[S43] Moore, M., Kaman, R., Klein, M., and Asundi, J., "Quantifying the value of architecture design decisions: lessons from the field," in *25th International Conference on Software Engineering, 2003. Proceedings.*, (Portland, OR), pp. 557–562, IEEE, May 2003.

[S44] Nord, R. L., Ozkaya, I., Kruchten, P., and Gonzalez-Rojas, M., "In Search of a Metric for Managing Architectural Technical Debt," in *2012 Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, pp. 91–100, IEEE, Aug. 2012.

[S45] Nord, R. L., Ozkaya, I., and Sangwan, R., "Making Architecture Visible to Improve Flow Management in Lean Software Development," *IEEE Software*, vol. 29, pp. 33–39, Sept. 2012.

[S46] Nord, R. L., Ozkaya, I., and Sangwan, R. S., "Analysis of Dependencies during Software Release Planning to Guide Architectural Decision Making Amid Competing Interests in Value and Cost," *Journal of Systems and Software*, pp. –, Sept. 2012.

[S47] Ojala, P., "Developing Value Assessment for SW Architecture," in *Seventh Working IEEE/IFIP Conference on Software Architecture (WICSA 2008)*, pp. 245–248, IEEE, Feb. 2008.

[S48] Oliveira Junior, E., Gimenes, I., Maldonado, J., Masiero, P., and Barroca, L., “Systematic evaluation of software product line architectures,” *Journal of Universal Computer Science*, vol. 19, no. 1, pp. 25–52, 2013.

[S49] Ozkaya, I., Kazman, R., and Klein, M., “Quality-Attribute Based Economic Valuation of Architectural Patterns,” in *2007 First International Workshop on the Economics of Software and Computation*, (Minneapolis, MN), pp. 5–5, IEEE, May 2007.

[S50] Schwanke, R., Xiao, L., and Cai, Y., “Measuring architecture quality by structure plus history analysis,” in *Proceedings - International Conference on Software Engineering*, pp. 891–900, 2013.

[S51] Ssaed, A. A. A., Wan Kadir, W. M. N., and Hashim, S. Z. M., “Cost benefits maximization using discount cost function for embedded system architecture optimization,” *International Journal of Software Engineering and its Applications*, vol. 6, no. 4, pp. 47–68, 2012.

[S52] Sullivan, K. J., “Software design: The options approach,” in *International Software Architecture Workshop, Proceedings, ISAW*, (New York, NY, United States), pp. 15–18, 1996.

[S53] Sullivan, K. J., Griswold, W. G., Cai, Y., and Hallen, B., “The structure and value of modularity in software design,” in *Proceedings of the ACM SIGSOFT Symposium on the Foundations of Software Engineering*, (Vienna), pp. 99–108, University of Virginia, 2001.

[S54] Sutcliffe, A., “Bridging users’ values and requirements to architecture,” in *2013 3rd International Workshop on the Twin Peaks of Requirements and Architecture, TwinPeaks 2013 - Proceedings*, pp. 15–21, 2013.

[S55] Sutcliffe, A., “The socio-economics of software architecture,” *Automated Software Engineering*, vol. 15, no. 3-4 SPEC. ISS., pp. 343–363, 2008.

[S56] Svahnberg, M., Wohlin, C., Lundberg, L., and Mattsson, M., “A Quality-Driven Decision-Support Method for Identifying Software Architecture Candidates,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 13, no. 05, pp. 547–573, 2003.

[S57] Wesselius, J. H., “Modeling architectural value: Cash flow, time and uncertainty,” *Lecture Notes in Computer Science (including subseries Lecture Notes* *in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3714 LNCS, pp. 89–95, 2005.

[S58] Yuan, S.-T. and Lu, M.-R., "An value-centric event driven model and architecture: A case study of adaptive complement of SOA for distributed care service delivery," *Expert Systems with Applications*, vol. 36, pp. 3671–3694, Mar. 2009.

C.2 Selected Studies in Systematic Mapping

[M1] Akbarinasaji, S., "Toward measuring defect debt and developing a recommender system for their prioritization," vol. 1469, pp. 15–20, 2015.

[M2] Alzaghoul, E. and Bahsoon, R., "Cloudmtd: Using real options to manage technical debt in cloud-based service selection," in *International Workshop on Managing Technical Debt (MTD)*, 2013.

[M3] Alzaghoul, E. and Bahsoon, R., "Evaluating technical debt in cloud-based architectures using real options," in *ASWEC 2014*, 2014.

[M4] Bohnet, J. and Döllner, J., "Monitoring code quality and development activity by software maps," in *International Workshop on Managing Technical Debt (MTD)*, 2011.

[M5] Brondum, J. and Zhu, L., "Visualising architectural dependencies," in *International Workshop on Managing Technical Debt (MTD)*, 2012.

[M6] Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., and Zazworka, N., "Managing technical debt in software-reliant systems," in *FSE/SDP Workshop*, 2010.

[M7] Buschmann, F., "To pay or not to pay technical debt," *Software, IEEE*, vol. 28, pp. 29–31, Nov 2011.

[M8] Cai, Y., Kazman, R., Silva, C. V., Xiao, L., and Chen, H.-M., "Chapter 6 - a decision-support system approach to economics-driven modularity evaluation," in *Economics-Driven Software Architecture*, 2014.

[M9] Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A., and Amanatidis, T., “Estimating the breaking point for technical debt,” in *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pp. 53–56, Oct 2015.

[M10] Codabux, Z. and Williams, B., “Managing technical debt: An industrial case study,” in *International Workshop on Managing Technical Debt (MTD)*, 2013.

[M11] Codabux, Z., Williams, B., and Niu, N., “A quality assurance approach to technical debt,” in *SERP*, 2014.

[M12] Curtis, B., Sappidi, J., and Szynkarski, A., “Estimating the principal of an application’s technical debt,” *Software, IEEE*, vol. 29, pp. 34–42, Nov 2012.

[M13] de Groot, J., Nugroho, A., Back, T., and Visser, J., “What is the value of your software?,” in *International Workshop on Managing Technical Debt (MTD)*, 2012.

[M14] Eliasson, U., Martini, A., Kaufmann, R., and Odeh, S., “Identifying and visualizing architectural debt and its efficiency interest in the automotive domain: A case study,” in *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pp. 33–40, Oct 2015.

[M15] Ernst, N. A., Bellomo, S., Ozkaya, I., Nord, R. L., and Gorton, I., “Measure it? manage it? ignore it? software practitioners and technical debt,” in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015, (New York, NY, USA), pp. 50–60, ACM, 2015.

[M16] Falessi, D. and Reichel, A., “Towards an open-source tool for measuring and visualizing the interest of technical debt,” in *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pp. 1–8, Oct 2015.

[M17] Falessi, D., Shaw, M., Shull, F., Mullen, K., and Keymind, M., “Practical considerations, challenges, and requirements of tool-support for managing technical debt,” in *International Workshop on Managing Technical Debt (MTD)*, 2013.

[M18] Falessi, D. and Voegele, A., “Validating and prioritizing quality rules for managing technical debt: An industrial case study,” in *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pp. 41–48, Oct 2015.

[M19] Falessi, D. and Kruchten, P., “Five reasons for including technical debt in the software engineering curriculum,” in *Proceedings of the 2015 European Conference on Software Architecture Workshops*, ECSAW ’15, (New York, NY, USA), pp. 28:1–28:4, ACM, 2015.

[M20] Fernández-Sánchez, C., Díaz, J., Pérez, J., and Garbajosa, J., “Guiding flexibility investment in agile architecting,” in *HICSS*, 2014.

[M21] Griffith, I., Taffahi, H., Izurieta, C., and Claudio, D., “A simulation study of practical methods for technical debt management in agile software development,” in *WSC 2014*, 2014.

[M22] Guo, Y., Spínola, R., and Seaman, C., “Exploring the costs of technical debt management a case study,” *Empirical Software Engineering*, 2014.

[M23] Guo, Y., Seaman, C., Gomes, R., Cavalcanti, A., Tonin, G., da Silva, F., Santos, A., and Siebra, C., “Tracking technical debt: An exploratory case study,” in *ICSM*, 2011.

[M24] Guo, Y. and Seaman, C., “A portfolio approach to technical debt management,” in *International Workshop on Managing Technical Debt (MTD)*, 2011.

[M25] Ho, J. and Ruhe, G., “When-to-release decisions in consideration of technical debt,” in *MTD workshop*, 2014.

[M26] Holvitie, J. and Leppnen, V., “Examining technical debt accumulation in software implementations,” *International Journal of Software Engineering and its Applications*, vol. 9, no. 6, pp. 109–124, 2015.

[M27] Izurieta, C., Rojas, G., and Griffith, I., “Preemptive management of model driven technical debt for improving software quality,” in *QoSA 2015*, 2015.

[M28] Kazman, R., Cai, Y., Mo, R., Feng, Q., Xiao, L., Haziyev, S., Fedak, V., and Shapochka, A., “A case study in locating the architectural roots of technical debt,” in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 2, pp. 179–188, May 2015.

[M29] Kruchten, P., Nord, R., and Ozkaya, I., “Technical debt: From metaphor to theory and practice,” *Software, IEEE*, vol. 29, pp. 18–21, Nov 2012.

[M30] Letouzey, J. and Ilkiewicz, M., “Managing technical debt with the sqale method,” *Software, IEEE*, vol. 29, pp. 44–51, Nov 2012.

[M31] Letouzey, J.-L., "The sqale method for evaluating technical debt," in *International Workshop on Managing Technical Debt (MTD)*, pp. 31–36, June 2012.

[M32] Li, Z., Liang, P., and Avgeriou, P., "Architectural technical debt identification based on architecture decisions and change scenarios," in *Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on*, pp. 65–74, May 2015.

[M33] Li, Z., Liang, P., and Avgeriou, P., "Chapter 9 - architectural debt management in value-oriented architecting," in *Economics-Driven Software Architecture* (Mistrik, I., Bahsoon, R., Kazman, R., and Zhang, Y., eds.), pp. 183 – 204, Boston: Morgan Kaufmann, 2014.

[M34] Li, Z., Liang, P., Avgeriou, P., Guelfi, N., and Ampatzoglou, A., "An empirical investigation of modularity metrics for indicating architectural technical debt," in *QoSA 2014*, 2014.

[M35] Lim, E., Taksande, N., and Seaman, C., "A balancing act: What software practitioners have to say about technical debt," *Software, IEEE*, vol. 29, pp. 22–27, Nov 2012.

[M36] Marinescu, R., "Assessing technical debt by identifying design flaws in software systems," *IBM Journal of Research and Development*, vol. 56, pp. 9:1–9:13, Sept 2012.

[M37] Martini, A., Bosch, J., and Chaudron, M., "Architecture technical debt: Understanding causes and a qualitative model," in *SEAA 2014*, 2014.

[M38] Mayr, A., Plosch, R., and Korner, C., "A benchmarking-based model for technical debt calculation," in *QSIC 2014*, 2014.

[M39] Mendes, T., Almeida, D., Alves, N., Spnola, R., Novais, R., and Mendona, M., "Visminertd: An open source tool to support the monitoring of the technical debt evolution using software visualization," vol. 2, pp. 457–462, 2015.

[M40] Naedele, M., Chen, H.-M., Kazman, R., Cai, Y., Xiao, L., and Silva, C. V., "Manufacturing execution systems: A vision for managing software development," *Journal of Systems and Software*, vol. 101, pp. 59 – 68, 2015.

[M41] Naedele, M., Kazman, R., and Cai, Y., "Making the case for a "manufacturing execution system" for software development," *Commun. ACM*, 2014.

[M42] Nord, R., Ozkaya, I., Kruchten, P., and Gonzalez-Rojas, M., "In search of a metric for managing architectural technical debt," in *WICSA/ECSA*, 2012.

[M43] Nugroho, A., Visser, J., and Kuipers, T., "An empirical model of technical debt and interest," in *International Workshop on Managing Technical Debt (MTD)*, 2011.

[M44] Oliveira, F., Goldman, A., and Santos, V., "Managing technical debt in software projects using scrum: An action research," in *Agile Conference (AGILE), 2015*, pp. 50–59, Aug 2015.

[M45] Power, K., "Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options," in *International Workshop on Managing Technical Debt (MTD)*, 2013.

[M46] Ramasubbu, N. and Kemerer, C., "Towards a model for optimizing technical debt in software products," in *International Workshop on Managing Technical Debt (MTD)*, 2013.

[M47] Ramasubbu, N. and Kemerer, C., "Managing technical debt in enterprise software packages," *Software Engineering, IEEE Transactions on*, vol. 40, pp. 758–772, Aug 2014.

[M48] Ramasubbu, N., Kemerer, C., and Woodard, C., "Managing technical debt: Insights from recent empirical evidence," *Software, IEEE*, vol. 32, pp. 22–25, Mar 2015.

[M49] Reimanis, D., Izurieta, C., Luhr, R., Xiao, L., Cai, Y., and Rudy, G., "A replication case study to measure the architectural quality of a commercial system," in *ESEM 2014*, 2014.

[M50] Schmid, K., "On the limits of the technical debt metaphor some guidance on going beyond," in *MTD workshop*, 2013.

[M51] Schmid, K., "A formal approach to technical debt decision making," in *QoSA*, 2013.

[M52] Schwanke, R., Xiao, L., and Cai, Y., "Measuring architecture quality by structure plus history analysis," in *ICSE*, 2013.

[M53] Seaman, C., Guo, Y., Zazworka, N., Shull, F., Izurieta, C., Cai, Y., and Vetro, A., "Using technical debt data in decision making: Potential decision approaches," in *MTD Workshop*, 2012.

[M54] Shull, F., Falessi, D., Seaman, C., Diep, M., and Layman, L., "Technical debt: Showing the way for better transfer of empirical results," in *Perspectives on the Future of Software Engineering* (Münch, J. and Schmid, K., eds.), pp. 179–190, Springer Berlin Heidelberg, 2013.

[M55] Siebra, C., Cavalcanti, A., Silva, F., Santos, A., and Gouveia, T., "Applying metrics to identify and monitor technical debt items during software evolution," in *ISSREW, 2014*, pp. 92–95, Nov 2014.

[M56] Singh, V., Snipes, W., and Kraft, N. A., "A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension," in *International Workshop on Managing Technical Debt (MTD)*, 2014.

[M57] Skourletopoulos, G., Bahsoon, R., Mavromoustakis, C., Mastorakis, G., and Pallis, E., "Predicting and quantifying the technical debt in cloud software engineering," in *CAMAD, 2014*, pp. 36–40, Dec 2014.

[M58] Skourletopoulos, G., Mavromoustakis, C. X., Mastorakis, G., Rodrigues, J. J. P. C., Chatzimisios, P., and Batalla, J. M., "A fluctuation-based modelling approach to quantification of the technical debt on mobile cloud-based service level," in *2015 IEEE Globecom Workshops (GC Wkshps)*, pp. 1–6, Dec 2015.

[M59] Sneed, H., "Dealing with technical debt in agile development projects," *Lecture Notes in Business Information Processing*, 2014.

[M60] Yli-Huumo, J., Maglyas, A., and Smolander, K., "The sources and approaches to management of technical debt: A case study of two product lines in a middle-size finnish software company," *Lecture Notes in Computer Science*, 2014.

[M61] Yli-Huumo, J., Maglyas, A., and Smolander, K., "How do software development teams manage technical debt? - an empirical study," *Journal of Systems and Software*, 2015.

[M62] Zazworka, N., Seaman, C., and Shull, F., "Prioritizing design debt investment opportunities," in *MTD workshop*, 2011.

[M63] Zazworka, N., Vetro, A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., and Shull, F., "Comparing four approaches for technical debt identification," *Software Quality Journal*, vol. 22, no. 3, pp. 403–426, 2014.