TECHNOLOGIES TO SUPPORT THE TECHNICAL DEBT MANAGEMENT IN SOFTWARE PROJECTS: A QUALITATIVE RESEARCH

Victor Machado da Silva

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Engenharia de Sistemas e Computação, COPPE, da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia de Sistemas e Computação.

Orientador: Guilherme Horta Travassos

Rio de Janeiro

Agosto de 2018

TECHNOLOGIES TO SUPPORT THE TECHNICAL DEBT MANAGEMENT IN SOFTWARE PROJECTS: A QUALITATIVE RESEARCH

Victor Machado da Silva

DISSERTAÇÃO SUBMETIDA AO CORPO DOCENTE DO INSTITUTO ALBERTO LUIZ COIMBRA DE PÓS-GRADUAÇÃO E PESQUISA DE ENGENHARIA (COPPE) DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE MESTRE EM CIÊNCIAS EM ENGENHARIA DE SISTEMAS E COMPUTAÇÃO.

Examinada por:

Prof. Guilherme Horta Travassos, D.Sc.

Prof. Rodrigo Oliveira Spínola, D.Sc.

Prof. Toacy Cavalcante de Oliveira, D.Sc.

RIO DE JANEIRO, RJ - BRASIL
AGOSTO DE 2018

Silva, Victor Machado da

Technologies to Support the Technical Debt Management in Software Projects: A Qualitative Research / Victor Machado da Silva – Rio de Janeiro: UFRJ/COPPE, 2018.

XIV, 183 p.: il.; 29,7 cm.

Orientador: Guilherme Horta Travassos

Dissertação (mestrado) – UFRJ/COPPE/Programa de Engenharia de Sistemas e Computação, 2018.

Referências Bibliográficas: p. 109-113.

1. Software Engineering. 2. Technical Debt. 3. Survey. I. Travassos, Guilherme Horta II. Universidade Federal do Rio de Janeiro, COPPE, Programa de Engenharia de Sistemas e Computação. III. Título.

Agradecimentos

Apesar de apenas meu nome constar na capa do trabalho, este não teria sido concluído sem a participação de inúmeras pessoas na minha vida.

Aos meus pais, Ana Cláudia e Clayton, por ensinarem que não se consegue nada na vida sem estudo e dedicação e por oferecerem apoio incondicional sempre. À minha irmã, Patrícia, pelo carinho e pelos incentivos. À minha esposa, Valnice, pela paciência e compreensão em todos os momentos, fáceis ou difíceis. Aos meus avós, Lídio, Nilda, João e Maria de Lourdes que foram e continuam sendo meus exemplos de vida. Aos meus tios, tias, primos e primas, que sempre apoiaram e comemoraram as vitórias, mesmo com toda a distância que normalmente temos que enfrentar. Ao Leonardo, pelas horas dedicadas na revisão do texto e pelas sugestões dadas.

Ao professor Guilherme, pelo direcionamento que recebi durante o mestrado, pela confiança depositada ao longo dos trabalhos, por ter me mostrado toda a importância da pesquisa no nosso dia-a-dia, pelas várias aulas ministradas, e por ter me ensinado a ser um melhor pesquisador, professor e profissional. Carregarei sempre comigo tudo o que eu aprendi.

Aos colegas que entraram comigo no mestrado na linha de Engenharia de Software, Diogo e Gláucia, e a todo o pessoal do grupo de Engenharia de Software Experimental (Breno, Jobson, Paulo, Thiago, Fábio, Luciana, Wladimir, Victor, Talita, Hilmer, Rebeca, Valéria, Bruno e Alessandro), pelo apoio, pela companhia e pelas inúmeras discussões e sugestões dadas no laboratório, nas aulas ou nas reuniões. Um agradecimento especial ao Helvio, pela enorme ajuda, pelas sugestões e pelas inúmeras revisões feitas durante as pesquisas e nos artigos desenvolvidos.

Aos professores Toacy Cavalcante e Rodrigo Spínola, por aceitarem participar da minha banca de defesa. A todos os professores da COPPE que tive o prazer de assistir às suas aulas (Claudson, Ana Regina, Toacy, Breno, Xexéo), e à toda a equipe técnico-administrativa da COPPE e da UFRJ que deram o suporte necessário durante todo o mestrado.

Agradeço, por fim, à CAPES pelo financiamento.

Resumo da Dissertação apresentada à COPPE/UFRJ como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

TECNOLOGIAS PARA APOIAR O GERENCIAMENTO DA DÍVIDA TÉCNICA EM PROJETOS DE SOFTWARE: UMA PESQUISA QUALITATIVA

Victor Machado da Silva

Agosto/2018

Orientador: Guilherme Horta Travassos
Programa: Engenharia de Sistemas e Computação

Introdução: Dívida técnica (DT) é um conceito relacionado a atributos de qualidade interna em projetos de software, como manutenibilidade. Apesar de existir um interesse pelo assunto entre pesquisadores, estudos indicam que o gerenciamento da DT (GDT) pela indústria ainda é incipiente. **Objetivo:** Este trabalho tem por objetivo produzir um resumo dos principais conceitos relacionados à DT, incluindo tecnologias atualmente disponíveis destinadas para o GDT. **Métodos:** Foi conduzido um *survey* com profissionais de organizações de software brasileiras (OSBs) para investigar o nível de conhecimento atual dos participantes sobre DT e GDT, incluindo tecnologias utilizadas pela indústria. Em seguida, foi desenvolvida uma *quasi*-revisão sistemática, com o propósito de levantar na literatura técnica tecnologias específicas para o GDT. **Resultados:** Foram observados indícios de que o conhecimento geral dos profissionais em OSBs com relação à DT e ao GDT é baixo, e apenas uma minoria adota práticas de GDT. Foram reunidas 99 tecnologias, entre ferramentas e práticas de GDT. Os resultados dos estudos foram sintetizados em *evidence briefings*, documentos de uma página destinados a aprimorar a transferência de conhecimento entre o meio acadêmico e a indústria. **Conclusões:** Esta dissertação contribui com a disponibilização de um pacote de pesquisa contendo o plano do *survey* e os questionários usados, como forma de facilitar o processo de replicação da pesquisa. Além disso, os *evidence briefings* desenvolvidos possuem aplicação direta na indústria, como ferramenta de auxílio à implantação de práticas de GDT nas OSBs.

Abstract of Dissertation presented to COPPE/UFRJ as partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

TECHNOLOGIES TO SUPPORT THE TECHNICAL DEBT MANAGEMENT IN SOFTWARE PROJECTS: A QUALITATIVE RESEARCH

Victor Machado da Silva

August/2018

Advisor: Guilherme Horta Travassos
Department: Computer Science and Systems Engineering

Background: Technical debt (TD) is a concept related to internal quality attributes in software projects, such as maintainability. Despite an increasing interest in this topic among researchers, studies indicate that the TD management (TDM) by software practitioners from the industry is still incipient. **Objective:** This work aims to produce a summary of the central concepts related to TD, including technologies currently available to manage the TD. **Methods:** A survey with practitioners from Brazilian software organizations (BSOs) was conducted to investigate the participants' knowledge level on TD and TDM, including technologies adopted by practitioners from the industry. Following this study, a *quasi*-systematic literature review was executed, with the purpose of gathering in the technical literature specific TDM technologies. **Results:** We observed indications that the general knowledge of the practitioners from BSOs regarding TD and TDM is low, and only a minority of the participants adopt TDM strategies. The results from the studies were synthesized in evidence briefings, one-page documents intended to improve the knowledge transfer between the researchers and the practitioners from the industry. **Conclusions:** This dissertation contributes with the distribution of a research package to aid in the replication process, if needed, containing the survey plan and the questionnaires used in the research. Moreover, the evidence briefings have a direct application by the practitioners from the industry, as a tool to aid the inclusion of TDM practices in BSOs.

INDEX

INDEX OF FIGURES

INDEX OF TABLES

1 Introduction

This chapter introduces the research, presenting the context, objectives and the methodology adopted. The dissertation organization is also presented at the end of the chapter.

1.1 Motivation and Context

Interest in software quality has increased significantly over the years in the software development community. Nowadays, software projects are becoming increasingly more complicated, as they must develop products to integrate several types of platforms and multiple hardware specifications.

Among the different aspects currently under discussion in the literature, the Technical Debt (TD) metaphor, developed in 1992, is gaining interest in several research groups. The current TD definition, proposed by Avgeriou et al. (2016), states that TD is "*a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents n actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability.*"

The financial industry inspires this metaphor. On the one hand, the debt acquired has an initial cost due to the effort required to solve the issue at that moment, called the debt principal. On the other hand, there is a growing additional cost due to all the software changes required for the project's evolution, called the interest.

As is the case in any financial debt, if a software incurs in TD and maintains it for a long time without taking any precautions or actions to reduce the debt, the added interest can overwhelm the project's budget, raising its total cost or even making it unfeasible.

The TD metaphor was rediscovered in the late 2000's by the Agile community, as an efficient instrument to facilitate the communication between technical and non-technical stakeholders. The shorter development cycles of a project conducted using Agile methodologies often lead to "shortcuts" in the teams' decision-making process to attend the client's deadlines. However, those "shortcuts" created a scenario in which the maintainability and evolvability of the software development project could be affected, fitting perfectly into the TD definition.

Another essential scenario where the TD metaphor is applicable is in startups. Usually, it is necessary to design sub-par solutions to achieve a better time-to-market, beating potential competitors in delivering a product to the final clients. Nonetheless,

the initial solution commonly delivered as an MVP (minimum viable product) often has serious issues regarding scalability properties, for instance, affecting the evolvability of the software. It leads to TD for the project.

These different scenarios indicate the need to develop the TD metaphor further, bringing other terms from the financial industry and improving previous known terms and definitions. For instance, the original TD metaphor was only associated with the source code, but it evolved to include other software artifacts.

TD management (TDM) is another area that has been receiving more attention over the past decades, as researchers and practitioners start to look for opportunities to improve on currently adopted practices to manage the TD. Alongside practices and strategies, practitioners are looking for tools and TDM-supporting technologies that are intuitive and easy to include in their development process. Despite their interest in these technologies, studies report that practitioners tend not to use TDM tools, either due to the lack of appropriate technologies for some TD types or due to the difficulty in inserting the current tools in their overall activities (ERNST et al., 2015).

The CAST report on application software health (CRASH), published in 2011 (SAPPIDI; CURTIS; SZYNKARSKI, 2012), evaluated an average TD of $3.61 per line-of-code (LOC). Out of the 745 applications sampled by the organization, 12% had over 1 million LOC, implying that those applications alone could be incurred at least $300 million in TD. In Java-EE applications, the average TD is even higher, at $5.42 per LOC.

If considered the global IT context, the numbers scale drastically. A Gartner study (2010) defined global "IT debt," as *"cost of clearing the backlog of maintenance that would be required to bring the corporate applications portfolio to a fully supported current release state."* Based on that definition, they evaluated the total amount of debt of $500 billion in 2010, reaching $1 trillion in 2015.

For Brazilian software organizations (BSOs), in particular, we have scarce information on how TD is managed. Only a small number of studies provide some information on TDM, such as (ASSUNCAO et al., 2015; OLIVEIRA; GOLDMAN; SANTOS, 2015; RIBEIRO; SPÍNOLA, 2016; SPÍNOLA et al., 2013; TONIN, 2018). Those studies, in general, tend to be restricted to a single organization and do not provide a broad spectrum of the TD reality in the country.

This global context of increasing TD software projects and the TDM-supporting tool demand by the software practitioners and the specific context of BSOs and the lack of information on how they manage the TD provided a potential research gap to be conducted in this dissertation, represented by the central research question of **Which technologies that support manage the technical debt in software projects could**

be adopted by Brazilian software organizations and their practitioners? This problem will serve as a guide for the activities and investigations conducted in this dissertation.

1.2 Objective

Given the context and the challenges previously presented, the primary purpose of this work is to aid BSOs in adopting technologies to support the TDM in their software projects. To achieve this purpose, this dissertation aims to **produce a summary of the central TD concepts and current technologies developed to manage the TD specifically**.

The Evidence Briefings (EBs), proposed by Cartaxo et al. (2016), are going to be adopted to summarize the acquired knowledge and provide means to transfer it from the academia to the Brazilian software industry. They should provide information in the one-page briefing, simplifying the knowledge transfer process and improving the likelihood of it being acquired and adopted by software practitioners.

The following specific objectives were defined to achieve the primary objective:

- Assess the current knowledge on TD and its management among practitioners from BSOs;
- Identify in the technical literature technologies created to manage the TD in software projects;
- Aggregate knowledge on TD, including its central concepts and definitions and the technologies to manage the TD, to serve as an instrument to transfer the knowledge to software practitioners in BSOs.

1.3 Methodology

The following methodology was adopted to conduct the research and to achieve the objectives presented previously. They are also presented in Figure 1.1 along with the studies timeline and evolution. Each methodology step is discussed in further details in the next chapters of this dissertation.

1. **Selection of the research topic:** the research's subject raised throughout two different masters' courses: "Continuous Software Engineering" and "Quality in Software Products," in which the TD metaphor was discussed in different situations and contexts, bringing interest among the participants. Those discussions supported the idea that this topic was appropriate to be further developed as part of a masters' dissertation.
2. ***Ad hoc* literature review:** after the topic was selected, an *ad hoc* literature review was conducted to obtain more detailed information on TD, its

properties, and management. During the review, it was observed that a considerable number of secondary studies were executed recently, mostly between 2014 and 2016, indicating that the research community is seriously considering this subject. Due to the number of secondary studies, it was decided that they could be used to support most of the claims and premises adopted in this work in the first moment. Further searches on more recent articles and studies were conducted to update the secondary studies.

3. **Survey research with practitioners:** the survey was designed and conducted using a formal protocol, according to guidelines obtained in the literature and discussed with details in Chapter 3 of this work. A survey was chosen as the primary research instrument to obtain the needed information directly from software practitioners since the scientific literature provided little knowledge on this particular aspect of TD. A questionnaire was developed and hosted online, and the results were collected and analyzed posteriorly.
4. **Literature review on TDM technologies:** to assess the currently available technologies to manage the TD, a *quasi*-systematic literature review (*q*SLR) was co-conducted, along with a doctorate student of the Experimental Software Engineering group (ESE). The decision to conduct the review with two researchers provided synergies in both individual researches and aided in reducing the researcher's bias that could have occurred if only one person was involved. The literature review was able to select the technologies and group them under several characteristics, like the type of evaluation and TD types covered by each technology.
5. **Aggregation of TD knowledge using Evidence Briefings:** to synthesize the information collected in both the survey and the *q*SLR, as well as the definitions and concepts provided by other studies (ALVES et al., 2016; LI; AVGERIOU; LIANG, 2015), the concept of EBs was used, generating three briefings discussing the TD definition, TD types, TDM activities and TDM technologies.
6. **Final report development:** the complete research was documented and organized in the dissertation format, to facilitate future references and provide more details on the findings.

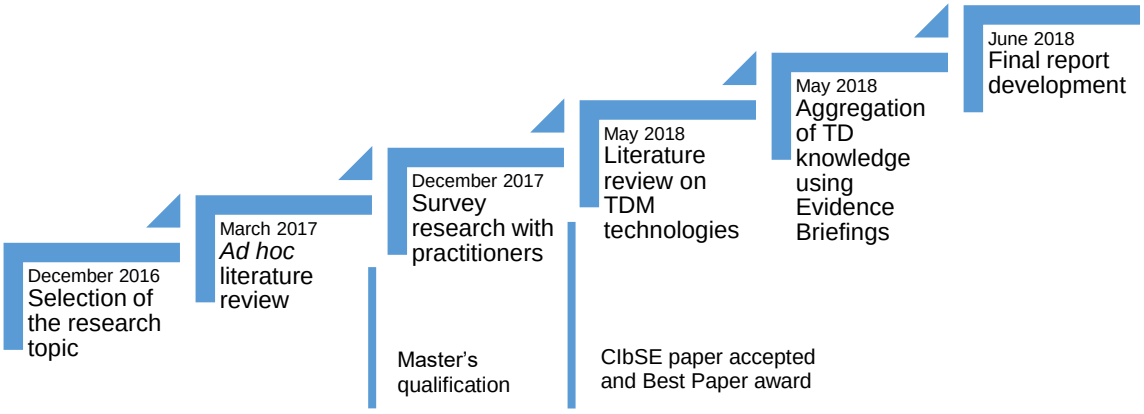

Figure 1.1 – Research timeline

The masters' qualification was presented on March 8th, 2017, including the initial literature review and the survey plan. The survey design and results were presented in the XXI Ibero-American Conference on Software Engineering, on April 24th, 2018, under the paper **A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil** (SILVA; JERONIMO; TRAVASSOS, 2018a). The paper received the Best Paper award of the ESELAW (Experimental Software Engineering Latin American Workshop) track. The award led to an invitation to extend the paper into a journal article for JSERD, to be submitted in October 2018.

An overview of this dissertation was presented in the paper **Technical Debt Management in Brazilian Software Organizations: A Need, an Expectation, or a Fact?** (SILVA; JERONIMO; TRAVASSOS, 2018b), accepted on the technical works track in August 2018 and to be presented in October 2018. Another paper is planned to be written to a journal (which one is yet to be defined), detailing the methodology and results from the literature review co-conducted by this author and presented in chapter 4.

1.4 Organization

This dissertation is organized into six chapters and four appendices. Besides this introduction, the following sections are presented from now on:

- **Chapter 2 – Technical Debt:** it introduces the TD definition and the metaphor evolution, from the first mention by Ward Cunningham (1993) to the most recent definition, proposed in the *Dagstuhl* seminar (AVGERIOU et al., 2016). The chapter then discusses the classification of TD, the financial concepts associated with the metaphor, the TDM activities and the current applications of TDM in the industry;

- **Chapter 3 – Survey research:** the survey plan, its execution, and results are presented in this chapter. It reports how the survey was designed, how the questionnaire was structured, validated and released, and presents the main findings and conclusions related to TD and TDM in the BSOs;
- **Chapter 4 – *Quasi*-Systematic Literature Review:** this chapter presents the literature review co-conducted alongside a doctorate candidate from the ESE group at COPPE/UFRJ. The research protocol is presented, including the selection criteria and the quality assessment of the studies;
- **Chapter 5 – Evidence Briefings:** to consolidate the knowledge gathered in the previous studies, this chapter presents the methodology executed to create the EBs;
- **Chapter 6 – Conclusion:** the main findings from both studies are revisited, including a further discussion on the answers to the research questions. The contributions and limitations of the research are also analyzed, and some future work opportunities are suggested;
- **Appendix A:** it presents the survey questionnaire to improve the replicability of this study. The questionnaire is supplied in its original language, Portuguese, and it was translated into English;
- **Appendix B:** it includes the tabulated results from the survey research, separating by the complete and incomplete sets of answers;
- **Appendix C:** here we present the information on the selected studies obtained from the *q*SLR conducted in chapter 4;
- **Appendix D:** finally, this appendix presents the final EBs, after the methodology adopted in chapter 5. As with the survey questionnaire, the EBs were written both in English and Portuguese.

2 Technical Debt

This chapter presents the information gathered through an ad hoc literature review regarding Technical Debt, its definition, general background, classification, management and the industry's perspective on the subject. Furthermore, a discussion of research gaps obtained from the literature review will be provided to ground the following steps of the research.

2.1 Introduction

A software project suffers a quality decrease as some essential artifacts are not produced, or tasks are poorly or not executed. The discussion among managers, architects, and developers regarding the decision on not performing these elements throughout the software project supports a better control over the project's weak spots, enabling the risk assessment and the next activities prioritization.

In this context, Ward Cunningham (1993) mentioned for the first time the idea of Technical Debt (TD), bringing a more straightforward and apparent financial analogy to the customers. In this metaphor, the release of a piece of code not attending some quality requirements or long-term necessities is like the acquisition of "debt" by the developer. Although the code successfully attends the current project demands, future occurrences might cause significant impacts due to the later alterations needed in the code.

From that point on, the TD metaphor was associated with the necessary effort (human or financial) to modify the unsatisfactory code. Other terms were also coined, eventually, increasing over time the scope of this new concept.

This chapter presents the evolution of the TD definition and details the main aspects involving the concept, like terms related to TD and its classification, TDM activities, and how the TD is being applied in the industry. Finally, we discuss the TD definition and the research gaps observed as a result of a technical literature review.

2.2 Evolution

The search for “technical debt” on the Scopus[1] database in February 2018 returned 340 documents (Figure 2.1), after excluding conference reviews and documents outside of the Software Engineer/Computer Science scope. The same search on the IEEE database[2] at the same period returned 230 documents. Besides the initial metaphor definition by Cunningham in 1992, the first mention observed in the technical literature was by Davis and Andersen (2009).

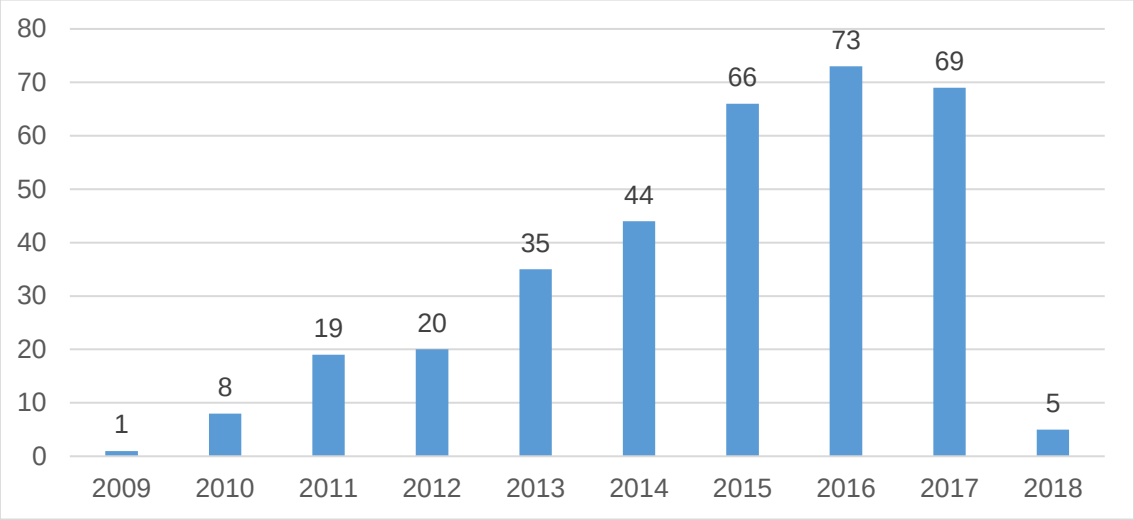

Figure 2.1 – Papers mentioning "technical debt" according to the Scopus database, distributed over the years

Initially, the TD metaphor was adopted mainly by the Agile community as a communication tool between technical and non-technical stakeholders (BROWN et al., 2010). Also, TD started to obtain some attention as the developers started to understand that some types of debt can be critical to the success of a software project (KTATA; LÉVESQUE, 2010).

Over the years many authors started to analyze the TD metaphor with more specific objectives, aiming to expand the concept and to better define a theory behind it (KRUCHTEN; NORD; OZKAYA, 2012; SEAMAN; GUO, 2011; TOM; AURUM; VIDGEN, 2013). TDM workshops were created to attempt consolidating the practical, mostly tacit knowledge and turn it into a solid theory, permitting the creation of new technologies to support the industry in managing TD.

Although reasonably disseminated, up until 2016 there was no precise definition of the TD concept, creating several inconsistencies in the technical literature (TOM;

[1] http://scopus.com

[2] http://ieeexplore.ieee.org/

AURUM; VIDGEN, 2013). Some definitions of TD over the years are *"a way to characterize the gap between the current state of a software system and some hypothesized 'ideal' state in which the system is optimally successful in a particular environment"* (BROWN et al., 2010), *"any side of the current system that is considered suboptimal from a technical perspective"* (KTATA; LÉVESQUE, 2010), and *"a tradeoff between implementing some piece of software in a robust and mature way (the 'right' way) and taking a shortcut which may provide short-term benefits, but which has long-term effects that may impede evolution and maintainability"* (KLINGER et al., 2011).

This imprecision on the TD definition could cause several misinterpretations and even a TD metaphor misuse and damage to the concept. Tom, Aurum and Vidgen (2013) affirmed that *"it is evident that the boundaries of technical debt, as reflected in academic literature, are fuzzy – they lack clarity and definition – and represent a barrier to efforts to model, quantify and manage technical debt."*

Throughout the years, there was some discussion on what should be considered TD and, mainly, what should not be considered it. Li et al. (2015) gathered in their systematic mapping study only six types of non-TD: defects, unimplemented features or functionalities, lack of supporting processes, unfinished tasks in the development process, minor code quality issues and low external quality. However, the authors state that this list has not been widely accepted by the research community, requiring more attention on the matter.

The lack of consensus on the TD definition was brought to attention during the *Dagstuhl* Seminar 16162, "Managing Technical Debt in Software Engineering" (AVGERIOU et al., 2016). This seminar gathered members from academia and industry to discuss many relevant points regarding the TD concept. At the end of the seminar the participants came up with a TD definition:

> *"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability."*

This definition expands what Cunningham (1993) proposed, including "design or implementation constructs." In other words, TD can incur in artifacts besides source-code, like requirements elicitation or process models. "Design", as indicated on the

definition, does not refer to the design phase of the development process, but to a broader scope of design, covering all the project phases, from conception to releasing and maintenance.

On the other hand, the *Dagstuhl's* definition narrows the initial concept, focusing TD on internal quality issues. This restriction is essential to differ TD from defects, as defects are associated with external quality issues, like reliability or usability.

As it can be observed, some differences between the TD definition and the first association with financial debt appeared throughout the years. The primary divergence is on the optionality to repay the TD item (GUO; SPÍNOLA; SEAMAN, 2016). However, some similarities remain. Similar to financial debt, strategical, controlled decisions that opt to postpone some tasks to obtain short-term gains such as shortening time-to-delivery can be decisive for a product's success (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016).

Nowadays, the definition proposed in the *Dagstuhl* Seminar is the most accepted among the researchers and will be adopted throughout this dissertation. This definition contradicts, though, with some previous concepts of what should be considered and what should not be considered TD. For instance, unfinished tasks in the development process are considered as a type of non-TD, as reported by Li et al.'s (2015) secondary study. However, it fits the *Dagstuhl's* definition of TD and should be considered as such in this dissertation.

It is important to mention that the concepts behind the TD definition are not exclusive to the Software Engineering domain. Being primarily a different perspective on the assessment of some kinds of risk, we can observe some TD-like events in other Engineering fields when, for example, structural engineers have to decide upon which type of analyses are going to be conducted to design a building. Although some analyses are necessary, maybe due to time constraints they decide not to conduct it, providing a potential future cost.

2.3 A financial metaphor

Along with the term "technical debt," several other financial terms were adopted by the software community to expand the concept further. Ampatzoglou et al. (2015) conducted a systematic literature review to gather which are the most common financial terms used in the context of TDM. They aggregated a TD financial glossary with the principal terms presented in the literature, which are reproduced below.

- **Asset:** 1. The effort that the development team saves by producing immature artifacts while accumulating TD. 2. Tools or approaches that lead to the decrease of TD;

- **Bankruptcy:** A software engineering project could be considered bankrupt, in case that it fails to survive/evolve (either being canceled or forced to be re-written from scratch) due to significant maintenance costs, caused by the accumulation of TD;
- **By-product:** An additional software product that can be produced with the effort that has been saved from producing immature artifacts while accumulating TD;
- **Financial leverage:** The ability of software companies to produce by-products, decrease time-to-market, etc. by accumulating TD;
- **Future value:** The value that an action (e.g., undertaking TD) performed now will have after a given period;
- **Interest:** The additional effort that is needed to be spent on maintaining the software, because of its decayed design-time quality;
- **Liability:** TD items, i.e., artifacts that have not been developed with optimal design-time quality and thus subject to improvement;
- **Present value:** The value that an action (e.g., a repayment activity) to be performed in the future has now;
- **Principal:** The effort that is required to address the difference between the current and the optimal level of design-time quality, in an immature software artifact or the complete software system (syn.: capital);
- **Repayment:** The amount of effort spent on improving design-time quality. This effort will decrease the effort needed for future maintenance tasks;
- **Risk:** The probability or threat that the TD items are accumulated in a design hotspot. Immature artifacts in such places of the system can hinder a product's viability;
- **ROI:** The ratio of (1) the additional amount of money that is earned by bringing the product earlier into the market or (2) the additional amount of money that has been earned from the company by investing the effort of the principal in an activity different than the improvement of design-time quality, over the principal;
- **Value-added:** The additional value that repayment of TD would bring to the software product.

2.4 Classification

Even before the academic interest in the TD metaphor, the industry already had presented alternatives to classify TD. Steve McConnell (2007) divided TD into two different types:

- **Unintentional TD:** This type of TD is not incurred with a strategical purpose. For example, a TD item could be a bad-written piece of code, created by an inexperienced programmer.
- **Intentional TD:** In this case, the TD is usually incurred with a strategical approach, when the team or the organization decides to achieve a short-term gain at the cost of a long-term effort. An example of intentional TD is the decision on developing a simplified architecture solution for the software, knowing that it might not attend the project's future needs. The intentional TD is further divided into two categories:
    - **Short-term debt:** TD that is expected to be paid off often. Usually, the project or organization has the resources to repay the TD item but has not at the current moment. An example of short-term debt is the delay of a specific feature to the next sprint if the team has no time available in the current sprint.
    - **Long-term debt:** TD that is taken strategically, with a more substantial time to be paid off. A long-term TD could be present when a specific requirement states that the database should support up to 1 million entries, but in initial stages the team decides to develop a solution that supports 100,000 entries, considering that this should be enough during the first two years of the system life.

Martin Fowler (2009) expanded the classification created by McConnell (2007), considering that, beyond the TD being intentional (deliberate) or unintentional (inadvertent), it can also be reckless or prudent. Those classifications are structured in the TD Quadrants, presented in Figure 2.2 and detailed from now on:

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

Figure 2.2 – Martin Fowler's (2009) TD Quadrants

- **Deliberate Debt:**
    - **Reckless:** This debt occurs when the team knows they are incurring in TD but does not make plans to repay it or decides to postpone this decision.
    - **Prudent:** According to the author, the TD item that is prudent and deliberate occurs *"because the team knows they are taking on a debt, and thus puts some thought as to whether the payoff for an earlier release is greater than the costs of paying it off."*
- **Inadvertent Debt:** Inadvertent debt is related to how the team deals with the TD item after they identify it.
    - **Reckless:** A reckless inadvertent debt occurs when the team, even knowing the TD presence, does not attempt to learn and/or eliminate the item.
    - **Prudent:** A prudent inadvertent debt is present when the team identifies the TD item and creates an action plan to repay it and, eventually, eliminate it.

Another perspective is to observe the original artifacts that the TD item was incurred or the TD item nature. Tom et al. (2013) named this classification scheme as dimensions of TD, naming five types of TD. Posteriorly, Li et al. (2015) conducted a systematic mapping study, expanding the TD dimensions into ten types. The most recent study attempting to classify the TD according to its nature or origin artifact, to our knowledge, is by Alves et al. (2016), in which the authors provide classification in fifteen TD types, listed from now on:

- **Design debt:** Associated mainly with violations of the principles of good object-oriented design, like extensive coupled classes;
- **Architecture debt:** Refers to problems related to the software architecture, such as violation of modularity;
- **Documentation debt:** Debt related to issues observed in the software documentation;
- **Test debt:** Debt found in testing activities, like planned tests that were not run;
- **Code debt:** Associated with problems found in the source code, that can make it harder to maintain, usually related to bad coding practices;
- **Defect debt:** Incurred when known defects have their elimination deferred to a later sprint. It is important to note a defect itself does not con-

stitute TD. The presence of a difference between the effort of eliminating the defect now and at a later sprint, however, can be related to TD;

- **Requirements debt:** Refers to decisions related to the software requirements, like the partial implementation of a given requirement, or the implementation of a functional requirement that does not attend all non-functional requirements;
- **Infrastructure debt:** Related to infrastructure issues that can impact negatively on some development activities, such as the delay of an upgrade or infrastructure fix;
- **People debt:** Debt associated with people issues that can impact negatively on some development activities, like tacit knowledge concentrated on only some people due to improper hiring or training activities;
- **Test automation debt:** Considered a subtype of test debt, it is related to the work involved in automating tests to enable continuous integration;
- **Process debt:** Debt associated with inefficient processes, or processes that are no longer adequate for the project;
- **Build debt:** Refers to issues that can hinder the build task, consuming unnecessary time;
- **Service debt:** Debt associated with inadequate selection of web services, leading to under- or overutilization of the system;
- **Usability debt:** Refers to inappropriate usability decisions that will need to be adjusted later, like the lack of usability standard;
- **Versioning debt:** Related to issues in the source code versioning, like unnecessary code forks.

Although practitioners adopt some of the TD types listed by Alves et al. (2016) in the industry, some of them had a small representation in their study. Ampatzoglou et al. (2016) conducted a study to understand how practitioners in organizations from the embedded systems domain perceive the TD. They performed an exploratory case study in seven organizations from four different countries. Among other research questions, the authors wanted to find what types of TD are more frequently occurring in embedded systems.

Their findings about the most frequent TD types in the practitioners point of view coincide with the taxonomy proposed by Alves et al. (2016), except regarding design debt, which is considered more relevant to researchers as it is for practitioners; and test debt and code debt, which seems to be more important to practitioners. Defect

debt, people debt, process debt, service debt and usability debt were not identified in the study.

2.5 Management

Li et al. (2015) state that TDM *"includes activities that prevent potential TD (both intentional and unintentional) from being incurred, as well as those activities that deal with the accumulated TD to make it visible and controllable, and to keep a balance between cost and value of the software project."* To our knowledge, their mapping study is the most recent on TDM activities, listing eight activities and the main approaches collected from the studies:

- **TD identification:** detects TD caused by technical decisions in software, either intentional or unintentional. The two most popular approaches are code analysis and dependency analysis;
- **TD measurement:** evaluates the cost/benefit relationship of known TD items in software or estimates the overall TD. Calculation model was the most common approach observed in the mapping study;
- **TD prioritization:** adopts predefined rules to rank known TD items, to support the decision-making process. The main approaches found to prioritize TD are cost/benefit analysis and "high remediation cost first" (repay TD items that cost more to resolve);
- **TD prevention:** establishes practices to avoid potential TD from being incurred. The most common approach collected by the mapping study is the development process improvement;
- **TD monitoring:** observes the evolution of known TD items over time. The mapping study gathered five approaches, all of them mentioned by only one study;
- **TD repayment:** eliminates or reduces the TD impact (principal and interest) in a software system. Refactoring is the most mentioned approach to repay TD;
- **TD representation/documentation:** represents and codes TD in a predefined standard, to address the stakeholders' concerns. All four studies collected in the mapping study propose an approach to represent the overall TD in a software system with TD items, or a unit of TD in said system. Usually, this representation includes fields like *ID*, *Location*, *Responsible*, *Type* and *Description*;

- **TD communication:** disclose the identified TD to the stakeholders. The most common approach collected by the mapping study is a TD dashboard, displaying TD items, types and amounts to inform all stakeholders.

2.5.1 Identifying, measuring, monitoring and documenting TD

Seaman and Guo (2011) discussed several approaches and techniques to measure and monitor the TD. They associate TDM with risk management in projects, where the risks are identified and described to improve the decision-making process to the stakeholders. Some of the risk attributes are used to the approaching development, especially impact and probability.

The proposed approach includes activities in TD identification, TD documentation, TD measurement, TD prioritization and TD monitoring. To identify TD, they adopt a TD list, which contains all identified TD items. They propose a template to include the primary information on the TD item, indicated in Table 2.1.

Every TD item should be measured according to three metrics: principal, interest probability and interest amount. Those metrics are estimated on a coarse-grain level by the developers, using values of high, medium or low. The authors consider it as sufficient for tracking the TD items and making preliminary decisions. However, it requires the following analysis on a fine-grain level, when there is more information on the TD items, and more detailed planning is needed.

Table 2.1 – TD item template (adapted from (SEAMAN; GUO, 2011))

ID	37
Date	March/31/2007
Responsible	Joe Blow
Type	Design
Location	Component X, especially function Y
Description	In the last release, function Y was added quickly, and the implementation violates some elements of the architectural design
Estimated principal	Medium (medium level of effort to modify)
Estimated interest amount	High (if X must be modified, this could cause lots of problems)
Estimated interest probability	Low (X not likely to be modified anytime soon)

With the TD in the software system adequately identified and measured, there are two courses of action:

- During the release planning, the team can use the TD list to evaluate which TD items should be paid first, according to predefined rules, like to repay TD items with the estimated principal “high,” or the ones with estimated interest probability “high”;

- During the project development, the TD list can be used to monitor the increase of TD independent of the release cycle, given that the team updates the estimated interest amounts and probabilities of each TD item.

2.5.2 Prioritizing TD

Ribeiro et al. (2016) conducted a mapping study to identify decision criteria to prioritize TD items. They collected fourteen criteria, categorized by the TD nature, customer, effort, and project. Those criteria were later evaluated by researchers and software practitioners, through a survey (RIBEIRO; SPÍNOLA, 2016). Nine criteria were evaluated, at the same time, as pertinent to support the decision-making process on TD prioritization and TD repayment, and relevant to the decision-making process. They are listed below, ranked from most to least pertinent:

- TD impact on the customer;
- TD severity;
- TD impact on the project;
- TD visibility;
- Analysis when the refactored part will be used;
- Nature of the project;
- TD location;
- Effort to implement the proposed correction;
- Cost/benefit.

Out of those nine criteria, “TD impact on the customer,” although answered as most pertinent by the survey participants, goes against the TD definition adopted in this work, in which it is associated with internal quality attributes, such as maintainability and evolvability.

However, this criterion can be considered pertinent to prioritize the TD incurred as a consequence of previous technical decisions that can currently affect the customer. For instance, a known defect was not eliminated in a previous iteration, but in the current iteration, the customer considers it crucial to be eliminated. It can lead to a situation where the criterion should be used on a prioritization decision. In this scenario, the TD elimination is a collateral effect that can impact the client, even if as an indirect consequence.

2.6 Applications in industry

The concept of TD is mostly a product of the industry’s necessities over time. Therefore, it is important to gather knowledge on how the industry applies the concept

to software projects. Several studies discuss or present the technologies involved in TDM activities, either approaches, tools or techniques. The following list indicates the main technologies identified in our literature review that are adopted by organizations worldwide:

- **TD identification:** Manual code inspection, SonarQube, CheckStyle, FindBugs (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016); CodeVizard and FindBugs (ZAZWORKA et al., 2013); SonarQube, Understand, CPPCheck, FindBugs, Sloccount (ERNST et al., 2015);
- **TD documentation:** TD template (SEAMAN; GUO, 2011); TD back-log/list, Documentation practice, JIRA, Wiki (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016); TD template (YLI-HUUMO et al., 2016)
- **TD communication:** TD meetings (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016); TD board (DOS SANTOS et al., 2013); Trello (OLIVEIRA; GOLDMAN; SANTOS, 2015);
- **TD measurement:** SonarQube, JIRA, Wiki (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016); TD evaluation template (YLI-HUUMO et al., 2016)
- **TD prioritization** Cost/benefit model, issue rating (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016);
- **TD repayment:** Redesigning, refactoring, rewriting (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016);
- **TD monitoring:** SonarQube, JIRA, Wiki (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016); Vtiger and JIRA (OLIVEIRA; GOLDMAN; SANTOS, 2015);
- **TD prevention:** Coding standards, code reviews, Definition of Done (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016);

Besides the technologies *per se*, there are some reports on the application of these technologies in the industry, indicating potential cases of success. The following studies analyze the industry's overall perspective on TDM activities in several countries.

2.6.1 Communication and Monitoring of TD

Dos Santos et al. (2013) reported the experience of an architecture team when adopting TDM practices during the first steps of using agile methodologies. They followed four steps:

- Training the team about what TD is and coming to a consensus that it should be managed;

- Based on their previous experience with Kanban, they developed a TD board, classifying several types of debt to show the maturity of each team in dealing with that specific type of debt;
- Quantifying the amount of debt using the Sonar tool (which accesses Code Debt); and
- Creating a virtual tile board containing valuable information (including the amount of debt obtained on Sonar) to get feedback on the TD evolution.

Among their findings, it is important to mention their concern on creating an organic culture by implementing a TD-oriented view without forcing it to the team. Also, the authors observed the communication improvement after implementing the board. Finally, the competition between teams resulted in an improvement of the TD repayment rate, achieving better results than expected.

2.6.2 Identification, Documentation, and Measurement of TD

Zazworka et al. (2013) evaluated the human elicitation of TD and compared it to automated identification. The study's participants were asked to report the TD items using Seaman and Guo's TD item template (2011) and then questioned about the difficulties found during the TD elicitation.

Parallel to that, the authors used the *CodeVizard* and *FindBugs* tools to identify the TD present automatically on the latest version of the subject project code. The results indicate that manual and automated identification tend to identify different TD items, with some overlapping. Additionally, different stakeholders identify different TD, indicating that the identification process should be conducted by several professionals on the team, to ensure better TD coverage in a project.

Finally, the study subjects took an average of 19 minutes to fill the TD template, stating that the fields principal, interest amount and interest probability were the most difficult to fill in.

Oliveira et al. (2015) conducted a similar study, this time an action research-based study with two Brazilian companies using Scrum. They had evidence that the TD was not managed in the companies' projects and decided to apply Seaman and Guo's TD item template (2011) to observe possible improvements.

Among their observations, they observed that all roles in the projects were able to identify TD, albeit professionals with different roles can find different types of TD. The average time to complete the identification part of the template was about three minutes.

The research participants suggested that the development team should be responsible for the debt measurement and prioritization since this role is more appropriate to derive the estimates. They also believe it is necessary to manually identify some of the TD types since the current tools were able to identify mostly design and defect debts automatically.

The authors reported some difficulty in filling three template fields associated with the TD measurement, i.e., calculating the principal and interest. One of the main reasons was the lack of historical data. This results coincided with Zazworka et al.'s (2013) work.

2.6.3 TDM maturity in software projects

Yli-Huumo et al. (2016) conducted a case study involving eight development teams of a single company from different projects across Europe. Their objective was to collect information and analyze how each team managed the TD. The content of the interviews was coded and classified into different TDM activities. For doing so, the authors used the mapping study provided by Li et al. (2015) that separated the TDM into eight activities, as discussed previously. Among the conclusions, the authors mentioned a lack of knowledge about the TD measurement, the absence of TD systematization and monitoring, and an interesting general structure for TD communication.

The authors also developed a framework, inspired by the software maturity models, for TDM, classifying the practices of each activity in: organized, received and not organized. Besides that, the framework indicates the professionals responsible for each activity and accessory practices or tools. This framework was adapted and is presented in Table 2.2.

Table 2.2 – TDM framework (adapted from (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016))

TDM levels/ TDM activity	Organized (Level 3)	Received (Level 2)	Unorganized (Level 1)	Responsible for activity	Practices/ tools for activity
Repayment	Continuous repayment with the monthly assigned percentage of the development tasks.	Repayment during typical tasks. Moreover, previously identified repayment tasks. Repayment based on current needs.	Repayment not conducted at all or only when it is not possible to avoid the issue any longer.	The development team, software architect(s)	Refactoring, redesigning, rewriting
Prevention	Mandatory prevention practices used by the team. Continuous practice during development.	Optional prevention practices. Not mandatory to use but recommended. Conducted based on current time constraints.	Prevention not assigned as part of the development practices. Conducted only occasionally.	The development team, software architect(s)	Coding standards, code reviews, Definition of Done
Representation/ documentation	Documentation is a mandatory practice in development. Issues are documented in a separate TD backlog.	Optional practice but recommended. Issues documented in a general development backlog without TD id.	Documentation not part of the development. Issues are left in developers' minds and notes.	The development team, software architect(s)	TD backlog/list, Documentation practice, project management tool (JIRA, Wiki)
Identification	Continuous identification conducted manually and/or with tools during development.	Identification optional during normal development. Conducted based on current time constraints.	Identification practices not assigned as part o the development. Only conducted when issues occur.	The development team, software architect(s)	Time reservation for manual code inspection. Code analysis tools (SonarQube, CheckStyle, FindBugs)
Measurement	Continuous measurement during development. Data analysis (various data used (e.g., quality, performance)). Assisted with tools.	Measurement of an optional practice. Measurement is done with simple data (number of TD issues) from development, and the data not necessarily used for other activities	Measurement not part of development practices.	Software architect(s), team manager	Data from management tools (SonarQube) and data from project management tools (JIRA, Wiki)
Monitoring	Continuous monitoring during development of various data. Tools used to support.	Monitoring based on pure data (number of TD issues). Conducted occasionally.	Monitoring not part of development practices.	Software architect(s), team manager	Monitoring tools (SonarQube). Project management tools (JIRA, Wiki)
Communication	Continuous discussions/meetings about TD issues with all the necessary stakeholders involved.	Discussions/Meetings organized only with some stakeholders.	TD not a topic in discussions/meetings and often handled only in coffee table discussions.	The development team, software architect(s), team manager	Specific TD meetings, TD included in discussion topics
Prioritization	Prioritization conducted continuously during development. Prioritization follows a specific method or model.	Prioritization based on hunches and estimations based on previous experiences. Prioritization is done merely without any specific model.	Prioritization not conducted, and decisions are made without reasoning or discussions.	Software architect(s), team manager	Cost/Benefit Model, Issue rating

2.6.4 TDM processes

Yli-Huumo et al. (2016) proposed a TDM process to conduct TD identification, TD documentation, TD measurement and TD prioritization. They executed action research in one of the largest IT companies in Scandinavia. The study includes five steps:

- **Step one:** Conduct interviews with the company's stakeholders to understand the current issues related to TD and its management. At this step, they found critical issues, like the absence of systematic processes for TD identification, evaluation or prioritization;
- **Step two:** Develop a process for TD identification, using previous knowledge of the product. The members of the product line conducted a manual identification since they did not have any automated tools to identify TD;
- **Step three:** Develop a process for documentation, adopting a TD backlog, with the TD items represented in a template similar to the one proposed by Seaman and Guo (2011), and indicated in Table 2.3. The TD types used in the documentation were gathered from Alves et al.'s study (2016);
- **Step four:** Organize a TD meeting (named "workshop" by the authors) to discuss and evaluate the TD items, using a specific template, adapted in Table 2.4;
- **Step five:** Sort the TD items in the TD backlog, obtaining a prioritization list to repay the TD items. The authors found that most of the TD items identified as design, architecture or code debt were assigned low priorities, while the ones identified as a test, test automation, and process debt were assigned a high priority.

Table 2.3 – Template for TD documentation (adapted from (YLI-HUUMO et al., 2016))

Field	Description
Technical Debt ID	Technical debt identification number
Date / Reporter	Reporting date / Reporter name
Technical debt name	Name of the identified technical debt
Description	Description of the identified technical debt
Alternatives	Explanation of possible alternative solutions
Rationale	Reasons to fix the technical debt

Table 2.4 – TD evaluation template (adapted from (YLI-HUUMO et al., 2016))

#	Question
1	What are the benefits of fixing this issue? (Business value, quality, productivity, fewer defects, etc.)
2	Are there any risks in fixing this issue? (It is expensive, breaks the system, etc.)
3	Why as this issue done previously like that?
4	How to fix this issue and what resources the fix would require?
5	From scale 1 – 5, how important would you rank this issue to be dealt? (1 – most important, 5 – not so important)

The authors concluded that TD could be brought visible adopting relatively simple practices and processes in the company, even if it does not have previous knowledge of TDM. Also, they observed that TD measurement is difficult to achieve at an excellent level, and therefore should be restricted to rough evaluations based on criteria like expert analysis.

2.7 Chapter considerations

The TD metaphor has a significant level of importance in the software development process. There is a consensus that all projects have some TD, as it often occurs involuntarily due to reasons such as the developer's lack of experience. Being intrinsically related to internal software quality aspects, whenever practitioners from a project face issues related to maintainability, for instance, they eventually have to deal with TD and its consequences.

TD, however, is not necessarily a bad thing in a software project. When properly managed and consciously incurred, TD can be a powerful tool in creating a new product, potentially granting the stakeholders advantages like a faster time-to-market or a better chance at attending deadlines required by the clients.

Although the number of TD-related studies is increasing every year at a fast pace, there is still little consensus on what should be considered or not TD. The *Dagstuhl* definition is considerable recent to have spread to the entire community and, specifically, to the industry. There is still a necessity to know if the software practitioners at the organizations indeed understand the TD concept as the academia defines it.

Furthermore, studies indicate that there is a lack of TDM supporting tools for the types that cannot be revealed in the source code (AMPATZOGLOU et al., 2016; LI; AVGERIOU; LIANG, 2015; OLIVEIRA; GOLDMAN; SANTOS, 2015; YLI-HUUMO et al., 2016; YLI-HUUMO; MAGLYAS; SMOLANDER, 2016; ZAZWORKA et al., 2013). It indicates possible research gaps, on developing new technologies to simplify the TDM through automated tools.

Additionally, despite most of the studies discussed in this chapter present a TD concept overview and state of the art regarding the activities and technologies that may be used to support its management, there is a need to observe the perception of TD and its management under the software industry point-of-view.

Finally, to our knowledge, there is scarce knowledge on how Brazilian software organizations (BSOs) manage the TD. Only a few studies were published reporting this matter (DOS SANTOS et al., 2013; GUO; SPÍNOLA; SEAMAN, 2016; ZAZWORKA et al., 2013). Therefore, it is essential to assess the knowledge level of the BSOs on TD and its management.

3 A Survey on the Software Industry Perception of Technical Debt and its Management in Brazil

It is presented in this chapter the development process of the survey research created to gather information on Technical Debt Management at Brazilian software organizations. It is discussed the related works on this matter, the survey plan, its design and the verification and validation process to support the survey release. Finally, the results of the survey are presented and discussed.

3.1 Introduction

In the previous chapter, some studies were observed through an *ad hoc* literature review, discussing the TD concept and technologies to support is management. However, only a few studies deal directly with the question of how software organizations perceive and apply the TD definition in their working environments (ALVES et al., 2016; LI; AVGERIOU; LIANG, 2015).

It is also important to mention that usually, the studies covering TD and its management focus only on some aspects of TD or on a limited population sample.

This context motivated us to investigate how practitioners from the Brazilian industry are currently adopting and managing TD. Although many studies cover this research topic, to our knowledge only Yli-Huumo et al. (2016) discussed it in a broader spectrum, and only with one organization. Therefore, there is a research gap on how the TD is perceived and managed in software organizations.

Furthermore, knowing that the country's culture, language influence the software development process, and beliefs (PRIKLADNICKI et al., 2007), there is still the question on if there is a difference on how TD is perceived and managed among organizations of different countries. Since, to our knowledge, there is no study investigating how BSOs perceive and manage the TD, we formulated the following central question of this study: **How BSOs, through their practitioners, perceive and manage the technical debt in software projects?**

A survey was designed and conducted with software practitioners engaged in BSOs to attempt answering this question. This chapter presents the results of this survey, intending to provide the following initial contributions:

- To get an initial perception of TD metaphor and its management in BSOs, using their engaged professionals as proxies;

- To make available a survey package with empirically evaluated instruments to support the gathering and aggregation of information regarding the TD perception and TDM activities, which can be tailored to other localities.

This chapter presents a background on surveys about technical debt conducted with software practitioners, followed by the survey design description and its results. Finally, the threats to validity are presented, and the survey results are discussed.

3.2 Related works

3.2.1 Other surveys on TD

Klinger et al. (2011) interviewed four software architects at IBM to obtain insights on how the organization perceives and manages TD. All four architects stated that the debt could incur unintentionally, showing up in the projects through, for example, acquisition, new alignment requirements or changes in the market ecosystem. They affirmed that unintentional debt is usually more problematic than intentional.

They also affirmed that the decision-making process on TDM is often informal and *ad hoc*. Finally, the interviewees claimed that there was a gap between executive and technical stakeholders, indicating a lack of a channel or common vocabulary to explain the TD to non-technical stakeholders.

Lim et al. (2012) conducted interviews with 35 practitioners with varying industry experiences from the USA. The authors aimed to understand how TD manifested in software projects and to determine which TD types were adopted by practitioners in the industry. They also investigated the causes, symptoms, and effects of TD, and finally, they questioned how practitioners deal with TD.

75 percent of the interviewees were not familiar with the TD metaphor, and the authors had to present a brief explanation of the concept. The participants described TD as tradeoffs between a short-term gain and an additional long-term effort. They affirmed that the effects of TD were not all negative, as the tradeoff depended on the product's value. Although they wanted a way to measure the TD, they claimed that measuring TD may not be that easy, as its impact is not uniform. Also, they claimed the key to measure TD is to evaluate the cumulative effect over time.

Finally, the authors suggested to start managing the TD in an organization through *"conducting audits with the entire development team to make technical debt visible and explicit; track it using a Wiki, backlog, or task board."*

Spínola et al. (2013) surveyed 37 practitioners from different countries (including Brazil), questioning which TD folklore statements, i.e., statements obtained online

and on published papers that *"might be subject to opinion, or that might be a good candidate for further investigation."*

The authors found a higher consensus between participants in two statements Participants strongly agreed on the statement "*if technical debt is not managed effectively, maintenance costs will increase at a rate that will eventually outrun the value it delivers to customers.*" They strongly disagreed with the statement "*all technical debt is intentional.*"

Ernst et al. (2015) executed a survey with 1,837 participants in three organizations in the United States and Europe. The authors found a *"widespread agreement on high-level aspects of the technical debt metaphor, including some popular financial extensions of the metaphor."* They also observed that the project context dramatically affects how the practitioners perceive TD. As they stated, only the software architecture was commonly seen as a source of TD, regardless of context.

65% of the respondents in this survey adopted only informal TDM practices. However, many respondents affirmed they manage TD through existing practices, such as risk processes or product backlog. 41% of the participants affirmed not to use any tool for managing TD, while only 16% use tools to identify TD. Additionally, participants affirmed that non-technical stakeholders were mostly unaware of TD in software projects.

Some participants installed some tools created specifically to manage TD, but they were rarely used, due to the configuration complexity or to the difficulty in understanding their results.

Rocha et al. (2017) surveyed with practitioners from BSOs to understand how the TD is dealt with in practice, at the code level only. Among their research questions, they investigated which are the factors that lead developers to create TD at the code level, and which practices can prevent developers to create TD ate the code level. 74 practitioners answered the survey, from which almost 72% affirmed to have medium to low/very low knowledge about the TD metaphor.

The participants affirmed that developers should follow the best programming practices to help prevent the TD, despite admitting they indeed contribute in creating TD on their projects. Among the main reasons to incur in TD, the participants answered management pressure, tight schedule, developer's inexperience, and work overload. Code review was pointed to as the most relevant practice to prevent the occurrence of TD.

Holvitie et al. (2018) conducted a multi-national survey to observe TD in practice, including practitioners from Finland, Brazil, and New Zealand. The authors opted to focus on practitioners that managed TD in organizations that adopt agile practices and

methodologies. 184 practitioners answered the survey. Approximately 20% of the participants had little to none knowledge on the TD definition.

35% of the Brazilian participants were able to provide an example of a TD instance. According to the study, the six leading causes of TD, selected by more than 50% of the participants, are inadequate architecture, inadequate structure, inadequate tests, inadequate documentation, software complexity and violation of best practices or style guides.

Finally, refactoring, coding standards, continuous integration and collective code ownership were perceived by most of the participants as having a positive effect on reducing the TD in software projects. 40-hour week journeys and open office space were practices assigned as having a neutral effect by roughly half of the participants. Regarding agile software development processes' and process artifacts, iteration reviews/retrospectives, iteration backlog, daily meetings, product backlog, iteration planning meetings and iterations were all assigned as having a positive impact on reducing TD.

3.2.2 Guidelines for conducting surveys in Software Engineering

Linâker et al. (2015) published a technical report providing guidelines for conducting surveys in Software Engineering. These guidelines were adopted throughout all this survey.

The authors indicate that the survey process consists of the following steps:

- Define the research objectives (already presented), describing the problem of interest and providing a context for the research questions;
- Identify the target audience, considering who can provide the information to achieve the research objective;
- Design the sampling plan, i.e., a plan containing how the members of the target population will be selected to participate in the survey;
- Design the survey instrument, usually a questionnaire;
- Evaluate the survey instrument;
- Submit the survey to the sample population;
- Analyze the survey data.

The next sections will provide further details on each step of the survey.

3.3 Research objectives

3.3.1 Global and specific objectives

This study's objectives were stated according to the goal-question-metric (GQM) paradigm (VAN SOLINGEN et al., 2002). The GQM paradigm is a hierarchical structure, in which the goals are defined, followed by the research questions, and then into the metrics. In other words, first, it is necessary to define the objectives of the research ("what?"), for only then assign the metrics ("how?"). The GQM provides ways to determine the variables that should be considered in an experiment, as those variables are based on the elicited goals previously defined.

The global objective of this study is to observe if the BSOs perceive, through their practitioners, the existence of TD and how the practitioners from the organizations conduct the different TDM activities.

The specific objectives are presented below:

- **Analyze** the TD and its management;
- **With the purpose of** characterizing;
- **With respect to** the level of knowledge and the adopted strategies, activities and technologies;
- **From the point of view of** software practitioners;
- **In the context of** Brazilian software organizations.

3.3.2 Research questions

The following research questions were formulated to achieve the global and the specific objectives of this study:

- **RQ1: Is there a consensus on the perception of TD among software practitioners?** Due to inconsistencies on the understanding of the TD concepts by software practitioners, reported in previous studies (ERNST et al., 2015; HOLVITIE et al., 2018; LIM; TAKSANDE; SEAMAN, 2012; SPÍNOLA et al., 2013), it is necessary to assess whether it is also observable in BSOs. Furthermore, it is necessary to evaluate if the understanding of TD by the practitioners is the same as it is being discussed in the academic environment. This research question can provide evidence to a universal understanding of TD between industry and academia.
- **RQ2: Do BSOs perceive the TD in their software projects?** Before the characterization of TDM activities conducted by the practitioners, it is essential to assess if the BSOs, through their practitioners, perceive the

existence of TD in their projects. By “perceive,” we mean to evaluate if the TD presence is observed and recognized as TD in their projects.

- **RQ2.1: Do BSOs manage the TD in their software projects?** If TD is perceived, then it is necessary to evaluate if the BSOs manage the TD in their software projects. This management does not need to be with formal processes and activities, as *ad hoc* management should also be worth noting. The answer to this question can aid the development of new support technologies, tailored to the maturity level of TDM in Brazil.
  - **RQ2.1.1: Which TD management activities are most relevant to software projects?** The goal of this question is to identify, among professionals, which TDM activities, among those proposed by Li et al. (LI; AVGERIOU; LIANG, 2015), are more critical to their practitioners in their projects. The answer to this research question could provide indications to further research on some less explored fields.
  - **RQ2.1.2: Which technologies and strategies are adopted for each TDM activity?** For all eight TDM activities proposed by Li et al. (LI; AVGERIOU; LIANG, 2015), this research question attempts to investigate which strategies and technologies are used to support each of them. There are already some studies indicating possible use of tools and other technologies in overseas projects (CODABUX et al., 2017; ERNST et al., 2015; GUAMAN et al., 2017a, 2017b; YLI-HUUMO et al., 2016). The purpose of this question is to evaluate if the BSOs are adopting the same technologies.

3.3.3 Questions this study cannot answer

This study focuses on managing the TD, so some characteristics about TD are considered as premises and are not questioned during the survey. Those are mainly connected to the TD classification, provided by Fowler (2009), Alves et al. (2016) and Li et al. (2015); the importance of TD on organizations (ERNST et al., 2015; KLINGER et al., 2011); most frequent TD types (AMPATZOGLOU et al., 2016); and most frequent TD sources (ALVES et al., 2016).

3.3.4 Open questions

The questions presented below cannot be answered by this survey and should be further observed in future investigations.

- How close to the *Dagstuhl* definition of TD (AVGERIOU et al., 2016) are understanding of TD by the practitioners from BSOs?
- Are the TDM technologies and strategies adopted by the BSOs effective and/or efficient?
- How much human or financial effort is saved by adopting the TDM technologies and strategies?
- According to the practitioners, what are the “best” TDM technologies and strategies?
- What is the most efficient way to estimate the impact of TD on software projects?

3.4 Target population and sampling

To achieve the research objectives and to answer the research questions, the practitioners from BSOs were selected as the target audience, while the population selected was the BSOs. The survey should include questions to reveal the participants’ job, their experience in software development and information on the organizations they work, such as size (comparable through the number of employees) and project domain to characterize the survey participants.

For this survey, the unit of observation is the software practitioner, while the unit of analysis is the software development projects. Finally, the search unit is the BSO.

The sampling design adopted is accidental, a non-probabilistic type of sampling, i.e., we cannot observe randomness on the selected units from the population. This decision can incur a threat to validity, which will be further discussed in Section 3.9. The only criterion adopted to select the units is convenience. An invitation to answer the survey was sent to a series of software development groups in the country, like RioSoft, Assespro, and Brascon. Other invitations were sent through the LinkedIn professional social network.

Finally, the survey was disclosed to the participants on three software-related events: RioInfo and SQBS, in Rio de Janeiro, and CBSoft, in Fortaleza. During the events, the researchers distributed flyers containing the survey title and a QR code with direct access to the online questionnaire (Figure 3.1). Posters were also designed and fixed in several locations on all events (Figure 3.2).

Figure 3.1 – Example of a flyer used to disclose the survey in Brazilian events

Figure 3.2 – Example of a poster used to disclose the survey in Brazilian events

3.5 Questionnaire design

The questionnaire was designed according to the guidelines presented in (LINÅKER et al., 2015). Due to geographical constraints, a web-based questionnaire was chosen, to attempt reaching professionals from other cities besides Rio de Janeiro (where the research was conducted).

The questionnaire is composed mostly of closed-ended questions. A small number of open-ended questions to get further information from the participant was necessary. It also adds partially closed-ended questions to deal with issues related to technologies and strategies for each TDM activity when the given options do not cover the entire possibility of answers from the participant.

Regarding the questionnaire structure, it is divided into fourteen sections, described from now on and summarized in Table 3.1. Table 3.2 presents the questionnaire extract, translated into English, offering some questions on TD identification. Each section starts with a brief explanation of its content and specific instructions. The *LimeSurvey* platform available at the Experimental Software Engineering (ESE) Group at COPPE/UFRJ supported the questionnaire implementation.

The questionnaire was configured to ensure the participant's anonymity. A welcome description was created to explain the survey importance for BSOs, as well as a summarized study structure description. The participants were asked to answer the questions based on their current (or most recent) software project and organization.

Each set of questions related to a specific TDM activity was conditioned to be provided to the participant only if s/he has some experience with that activity to avoid the problem of lengthy survey questionnaires (that can reduce the response rate severely). For example, if a participant is experienced only with TD identification and TD communication, only these two sections should be answered, while the other six sections would be hidden from the participant.

Also, other conditional sections in the questionnaire were defined as whether the participant is not familiar with the TD concept and whether the organization or the project do not apply any TDM activity. In both cases, the participant will not be able to answer the following questions, which focus explicitly on TDM activities. This decision reduces the survey length to the participant, potentially increasing the response rate (LINÂKER et al., 2015).

Table 3.1 – Questionnaire sections

Sections	Topic	Description
1	Participant characterization	Obtain personal information regarding the participant, such as experience, degrees, among others.
2	Organization characterization	Gather information about the organization the participant works or has worked recently.
3	Project characterization	Obtain information about the project that will be considered by the participant in the survey.
4	TD perception	Collect information on the participant knowledge regarding TD, including what can be considered TD. Also, determine if the organization and the project s/he works have strategies for TDM.
5	TDM (general)	Ask the participant on which TDM activities are adopted on the project s/he works. Gather responsibilities for each activity and gather an idea of the importance of each activity from the participant's point-of-view.
6 – 13	TDM (activities)	Gather information on several aspects regarding each one of the TDM activities proposed by (LI; AVGERIOU; LIANG, 2015).
14	TDM (other)	Provide space to the participant to describe other activities that are executed in the organization.

Table 3.2 – Survey – Extract of the TD identification section

Question	Answer options
Is there a formal strategy to identify TD?	( ) Yes, we have a formal procedure to identify TD. ( ) No, the identification of the TD is executed only informally.
Should all stakeholders apply the TD identification strategy?	( ) Yes, the strategy is mandatory for all stakeholders. ( ) No, the strategy is considered only a suggestion.
In which moment of the Project is the TD identified?	( ) There is no defined period; we identify the TD whenever we perceive some issue. ( ) We always identify the TD at the end of each iteration/sprint. ( ) The TD identification is continuous, i.e., occurs throughout the development process.
Mark below all tools or techniques that are used to identify TD.	[ ] Manual coding inspection [ ] Dependency analysis [ ] Checklist [ ] SonarQube/SQALE [ ] CheckStyle [ ] FindBugs [ ] CodeVizard [ ] CLIO [ ] Other (cite which)

3.5.1 Characterization sections

These three sections include questions regarding the participants' role in the projects, their academic formation, their working experience in software projects, the field of their organization, its size and if it has any maturity model evaluation in software processes.

To assess the size of the organizations, we adopted the SEBRAE/IBGE classification of organizations, consisting in micro (fewer than ten employees), small (between 10 and 49), medium (between 50 and 99) and large (more than 100) organizations. Although this grouping does not constitute a world-level standard, it attends the primary necessity for this study, which is a means to estimate the total number of organizations represented by the participants that answered the survey.

Finally, the projects in which the participants work are also characterized, through their domain problem and their lifecycle model. In this last question, the agile software development method was included for simplification purposes, even though it does not characterize as a lifecycle model.

3.5.2 TD perception

This section aims to gather the general participant understanding, regarding the TD definition and its overall aspects. The first question in this section is whether the participant knows what TD means. It was not our purpose to inquiry participants that do not know the meaning of TD, as they can provide wrong answers on the TDM sections.

Thus, the participants that do not know the TD definition should finish their questionnaire in this question.

To the participants that claim they know TD, a follow-up question was designed, to assess which common issues in software development should be considered TD. Those issues were gathered mainly from Li et al. (2015) for non-TD issues. The issues associated with the TD definition were obtained throughout an *ad hoc* literature review.

Following the general understanding of TD by the participants, they were questioned if TD was perceived in their most recent project, i.e., if they could notice any issues that could be associated with TD. An affirmative answer on this question permits the participants to answer two follow-up questions: if their organization adopts any TDM activity and if their manager (or themselves) adopt any TDM activity, regardless their organization adopting any.

If the participants answer “yes” to any of those two questions, then they can answer the remaining questionnaire.

3.5.3 TD management

The purpose of this section is to identify the adoption and relevance of TDM practices in the BSOs’ projects. The participants were clarified that by “technical debt management,” they should consider all activities that organize, monitor and control the TD and its impacts on software projects.

The participants were then asked to select which TDM activities were conducted in their projects, based on the list of TDM activities provided by Li et al. (2015). An additional option was included to provide space for the participants so that they can mention other TDM activities not discussed in Li et al. studies.

For each TDM activity selected by the participants, an additional question was created to ask which roles were responsible for conducting that activity. The leading roles offered as answers were obtained from Yli-Huumo et al.’s study (2016), but the questionnaire provided an open-ended question, so the participants could elaborate in case another role should be considered responsible for that activity.

3.5.4 Sections concerned with specific TDM activities

Eight sections follow the questionnaire, asking information regarding each one of the TDM activities proposed by Li et al. (2015). They are only available to the participants if they select those activities in the previous section.

At the beginning of each section, the TDM activity is described, to improve the participants’ knowledge and reduce the probability of misunderstanding of the proposed questions.

Those sections follow the structure presented below, for each activity. The list of tools and techniques in each section is obtained mainly from Li et al. (2015) and Yli-Huumo et al. (2016), but other tools or techniques may be added throughout the revision process.

3.5.4.1 TD identification

The participants were asked if there is any formal approach to identify TD. If there is a formal approach, the participants were asked if it is mandatory to all the people involved in that activity, or if it is only optional. Next, they were asked when the TD is identified, followed by which tools or techniques are used to conduct that activity.

Particularly for TD identification, a subsection was created to assess if and how the TD is classified after it has been identified. The classification methods were the ones presented in chapter 2 of this dissertation. A follow-up question was created, in case the participants answered they classify TD according to its type. In this case, TD types proposed by Alves et al. (2016) were mentioned, providing space for the participants suggest any other TD type.

3.5.4.2 TD documentation/representation

The participants were asked if there is a standard to follow when documenting TD, and if it is mandatory to all of those involved in this activity. Then they were asked how the TD items are documented or cataloged, followed by the tools or practices adopted to document TD.

3.5.4.3 TD communication

For TD communication, the participants were only asked how the unresolved TD items are communicated between the project stakeholders, and which tools or practices are used to the TD communication.

3.5.4.4 TD measurement

The participants were asked if there is any strategy previously defined to measure TD, and how it is measured efficiently. They were asked which information or variables are used to measure the TD items. Finally, they were asked which tools or practices are used to support the TD measurement.

3.5.4.5 TD prioritization

For TD prioritization, the participants were asked how the TD is prioritized, and which tools or practices are adopted during the activity. Finally, they are asked which criteria are used to support the TD prioritization, based on the list proposed by Ribeiro et al. (2016), presented below:

- Debt items that impact directly on the customer;
- Debt items with a high level of severity;
- Debt items that offer the most significant impact on the project;
- Debt items of critical projects;
- Debt items that can be perceived by the user;
- Debt items that are in widely used parts of the systems;
- Debt items located in a resource that will change due to a development or maintenance activity;
- Debt items that require less effort to be paid;
- Debt items with good cost-benefit.

Only two of the criteria presented above were not included in the final survey, after some discussion: “debt items that offer the greatest impact on the project”, due to its similarity in outcomes with “debt items with high level of severity”; and “debt items that can be perceived by the user”, since it contradicts with the TD definition adopted in this work, that it is mostly an internal quality attribute, therefore not visible to the user.

3.5.4.6 TD repayment

The participants were asked if there is any planning to repay TD, and then which techniques are adopted for the TD repayment.

3.5.4.7 TD monitoring

Like the TD repayment, the participants were asked how the TD is monitored, and which tools or practices are used to monitor it.

3.5.4.8 TD prevention

For this section, the participants were asked if there are any formal practices conducted to prevent the TD, and whether they are mandatory or optional to the stake-holders. Finally, they were asked about the tools or practices adopted to prevent the TD.

3.5.4.9 Other TDM activities

One last section is provided to gather information regarding other TDM activities that eventually are not on the list proposed by Li et al. (LI; AVGERIOU; LIANG, 2015) and yet are used in the participant’s software organization. The participants were asked if there is any formal approach to conduct these activities if they are mandatory to the stakeholders, and when they are conducted during the development process.

Finally, two open-ended questions are given, to ask which tools or techniques are adopted, and to ask for additional information on those activities.

3.5.5 Survey evaluation

The survey questionnaire passed through three types of evaluation:

3.5.5.1 Format and content revision

The first step, after the development of the main questionnaire, was to evaluate it with the advisor and another researcher from the ESE Group, in which this research is being conducted. Whenever a comment or improvement point was observed, the questionnaire was revised and submitted one more time for evaluation.

After three evaluation cycles, the questionnaire was submitted to the pilot trials.

3.5.5.2 Pilot trials

According to Linåker et al. (2015), a pilot trial is conducted using the same artifacts and procedures designed for the final survey, including the survey questionnaire and the execution method, but with a small number of participants from the target population. Our pilot trials were executed in June and July 2017.

Seven practitioners were invited to the pilot trials, being five on the first cycle and two on the second. Five of them work with software projects and comes from the ESE Group. The other two participants also work on software projects but are from outside the research group. All of them have some prior experience with TD and/or TDM, mostly in the industry.

The invitation was sent by e-mail, with the main instructions and the questionnaire link. They were asked to answer the questionnaire and return their feedback regarding, but not restricted to:

- Total response time;
- Question chaining, i.e., whether the questions have a logical connection between them;
- A proper understanding of questions and answer options;
- Answers completeness;
- Presence of ambiguity on questions and answers; and
- Clarity of the instructions.

All pilot participants answered the pilot survey within a week. The average answering time was 15.2 minutes. The most relevant comments were associated with:

- Usability issues, due to the *LimeSurvey*'s default configurations;
- Clarity of questions and options for answers;
- Suggestions to further detail the differences in TD activities to avoid misunderstandings; and

- Suggestions to define the necessary questions to avoid incomplete responses that could invalidate the questionnaire.

These issues were later discussed internally, and modifications were applied to the final questionnaire. Overall, we did not observe significant comments or doubts about either the answer options or the questions descriptions, suggesting that the questionnaire was consistent to be used in the study.

3.5.5.3 Final evaluation from pilot trials participants

After the final revision of the questionnaire, the survey was submitted one more time to the pilot trials participants, to confirm that the modifications improved the issues raised on their comments.

3.6 Survey release and data collection

After the pilot trials, the final survey was released and was available from July 2017 to June 2018. As previously mentioned, the sampling technique was by convenience, using professional social networks databases, contacts in BSOs from the researchers involved in the study and participation in Brazilian workshops and conferences during the survey execution time. The target audience is the practitioners engaged in BSOs. The questionnaire in English and Portuguese versions are available in Appendix A.

3.7 Survey results

In total, 62 participants answered the survey, with 36 complete answers. Four participants (ids. 37, 41, 61 and 86 on Appendix B) did not complete the survey but reached the questionnaire's section 4, regarding TD perception. Thus, they were included in this initial analysis. The remaining 22 incomplete responses were not included in the analysis. Figure 3.4 summarizes the survey responses.

3.7.1 Participants' characterization

The respondents from the 40 valid answers have an average of 14.15 years of working experience in software projects. Only four respondents reported having an incomplete undergraduate degree, while the remaining 36 respondents hold at least an undergraduate degree. Out of those 36, 26 participants reported holding either a specialization degree, a master or a doctorate (Figure 3.3). The two most recurring roles stated by the participants (Figure 3.5) are project manager (14) and programmer (9).

Regarding the BSOs in which the respondents work (Figure 3.6), most of them (23) are from the Information Technology (IT) sector, followed by Government (7), Consulting, Telecommunications, Financing (2 participants each), Education, Media,

Oil & Gas and Software Engineering (1 participant each). Referring to the project development (Figure 3.10), most of them (35) adopt agile or incremental lifecycle models. Two projects adopt the spiral model, while three adopt the waterfall model.

Due to the questionnaire anonymity, it is not possible to precise the number of organizations represented in the survey. However, it is possible to estimate it through the organization's and project's characterization. For example, the organization in which participant with id. 21 works are from the IT sector, has more than 100 employees and the project he/she works has the government problem domain, developed adopting the agile methodology. It is possible to obtain 12 different combinations for the organizations, and 30 different combinations for the projects using this information from all participants.

Figure 3.7 to Figure 3.9 provide more information about the characterization section.

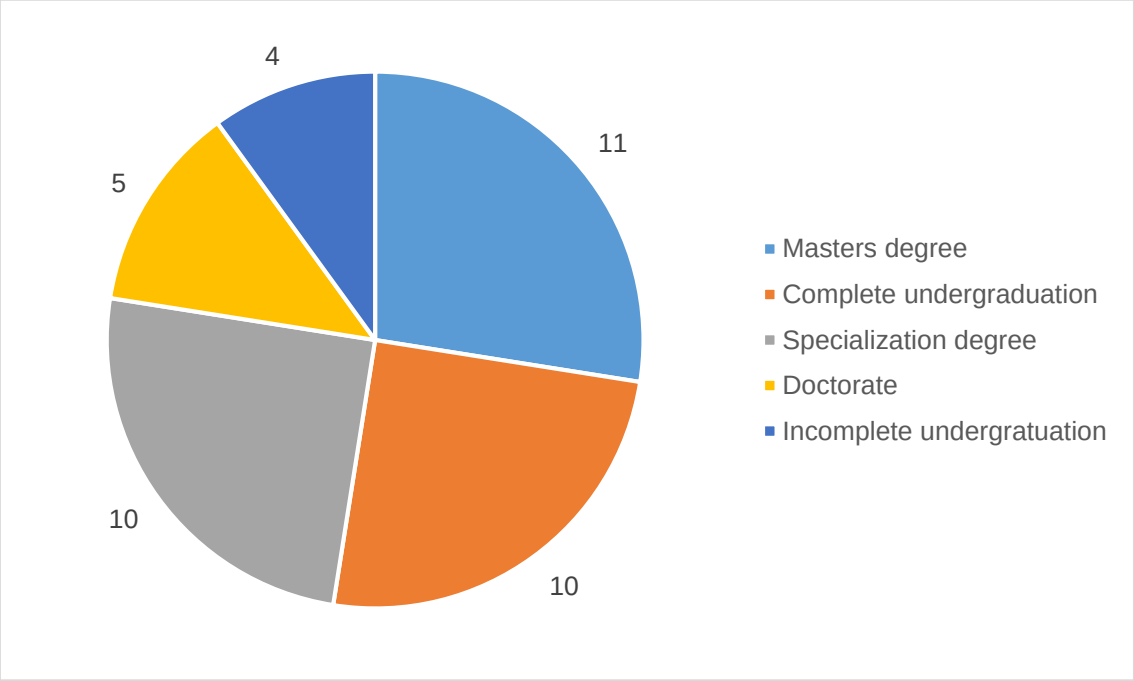

Figure 3.3 – Academic formation

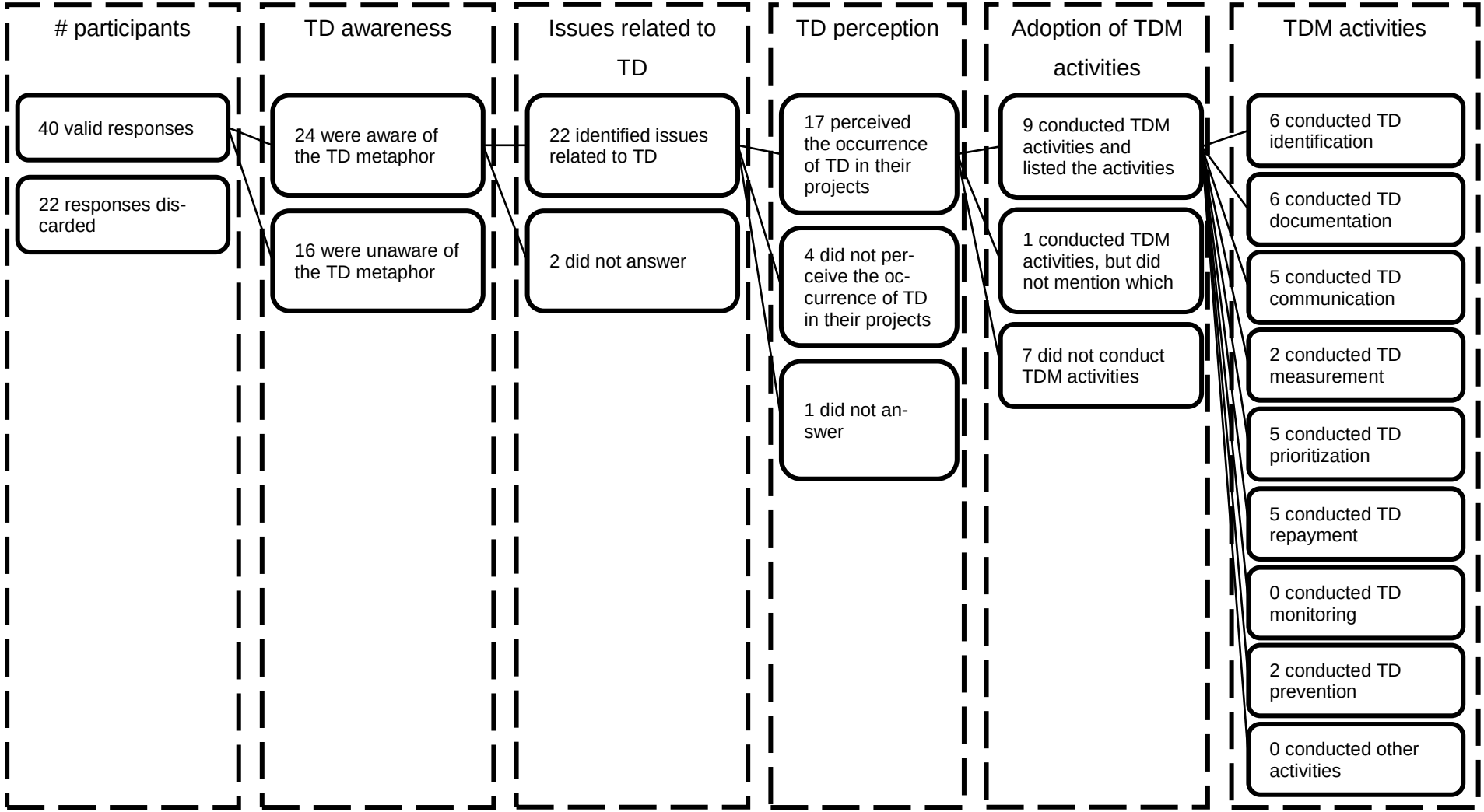

Figure 3.4 – Summary of the survey responses

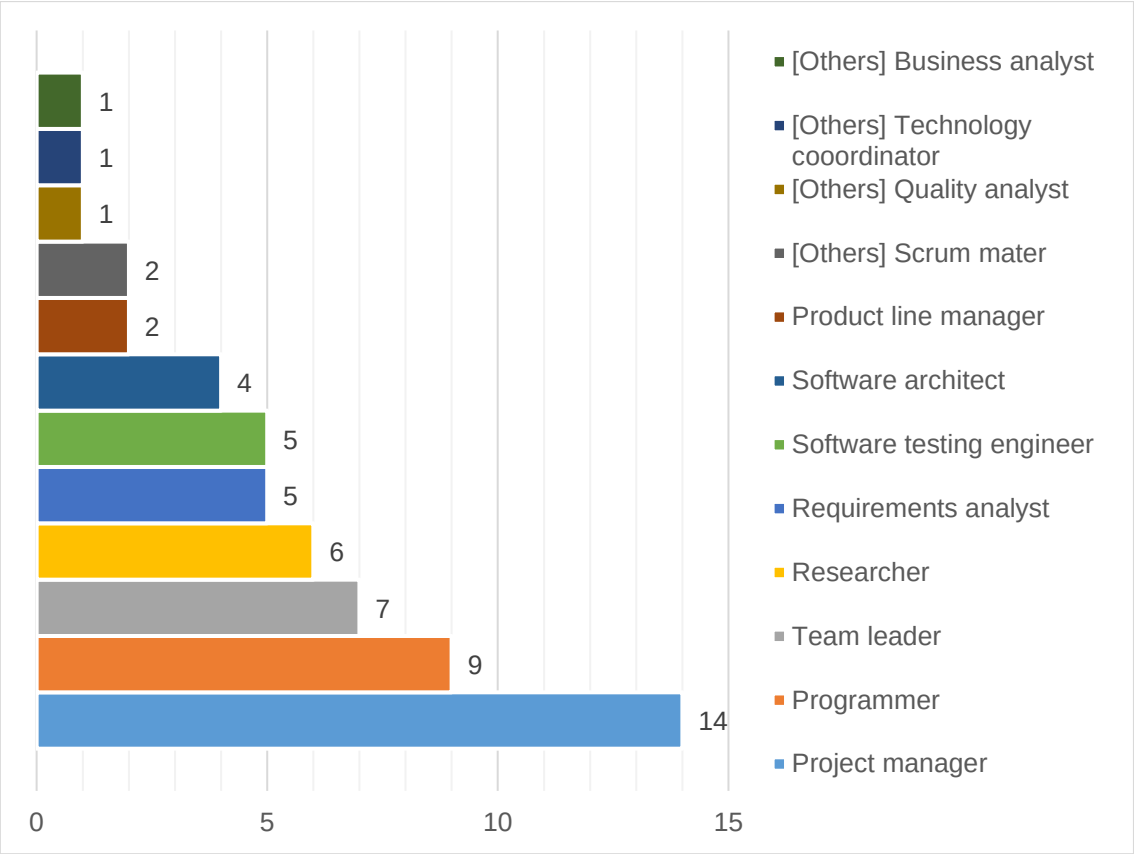

Figure 3.5 – Participants' role in the organizations

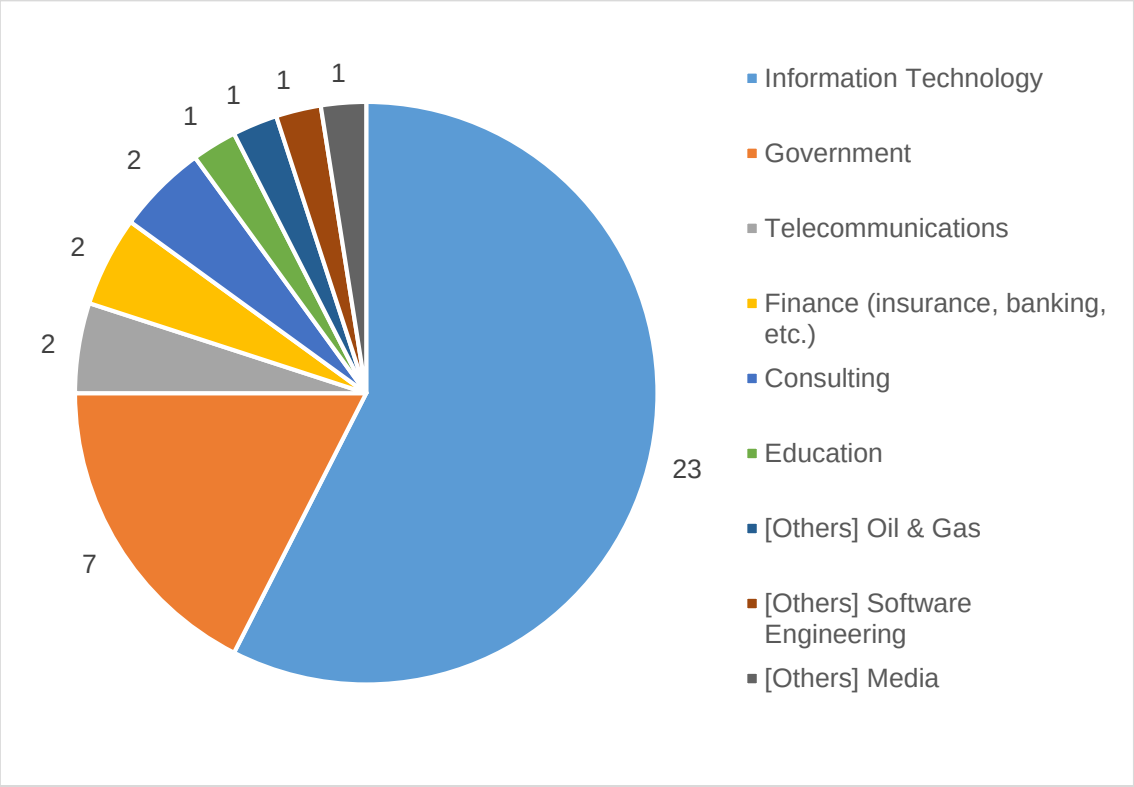

Figure 3.6 – Organizations' sectors

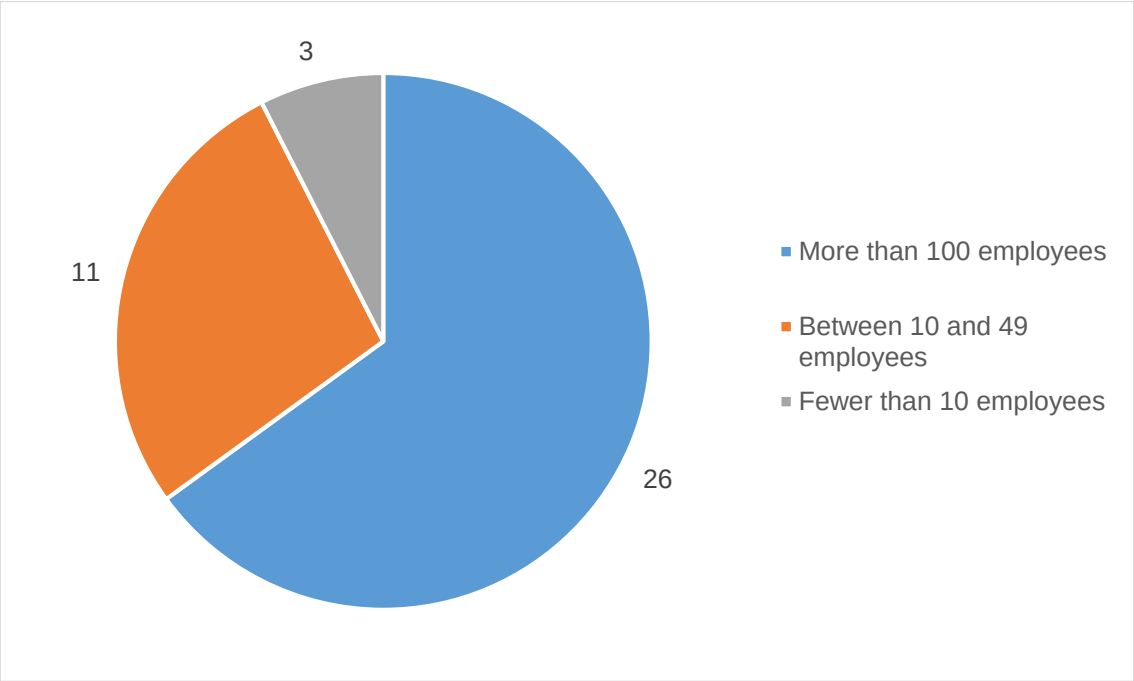

Figure 3.7 – Size of organizations, in the number of employees

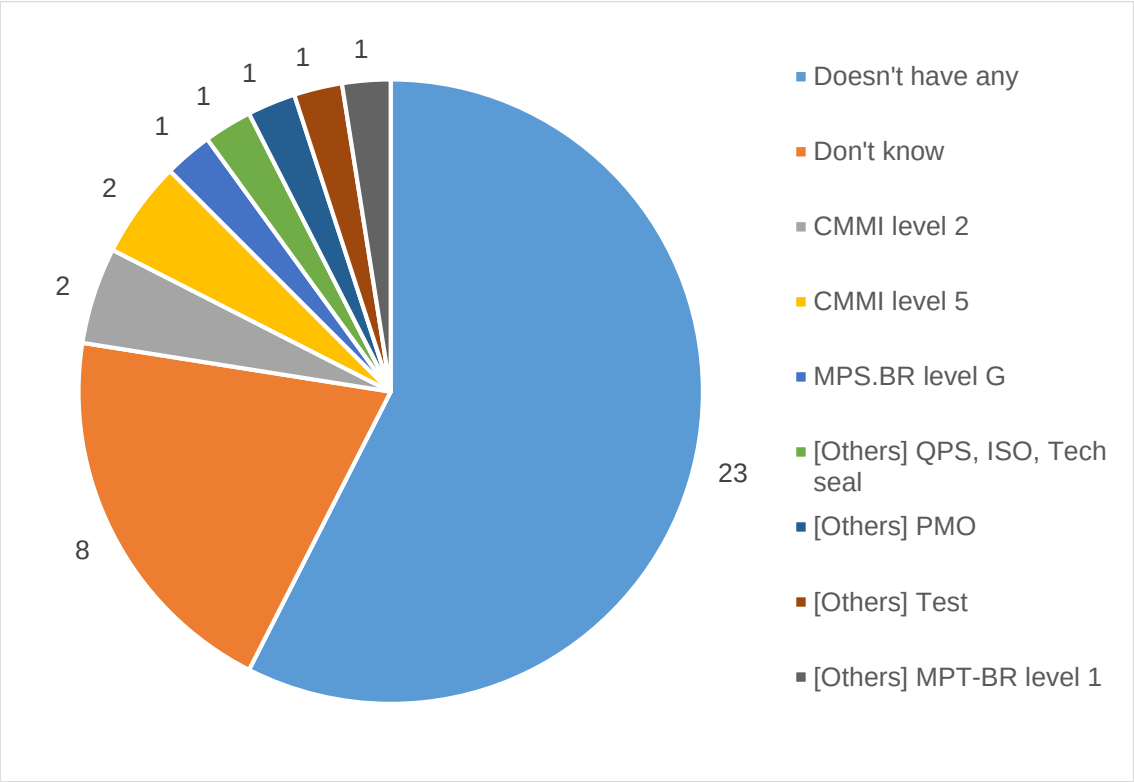

Figure 3.8 – Evaluation of the maturity level of organizations

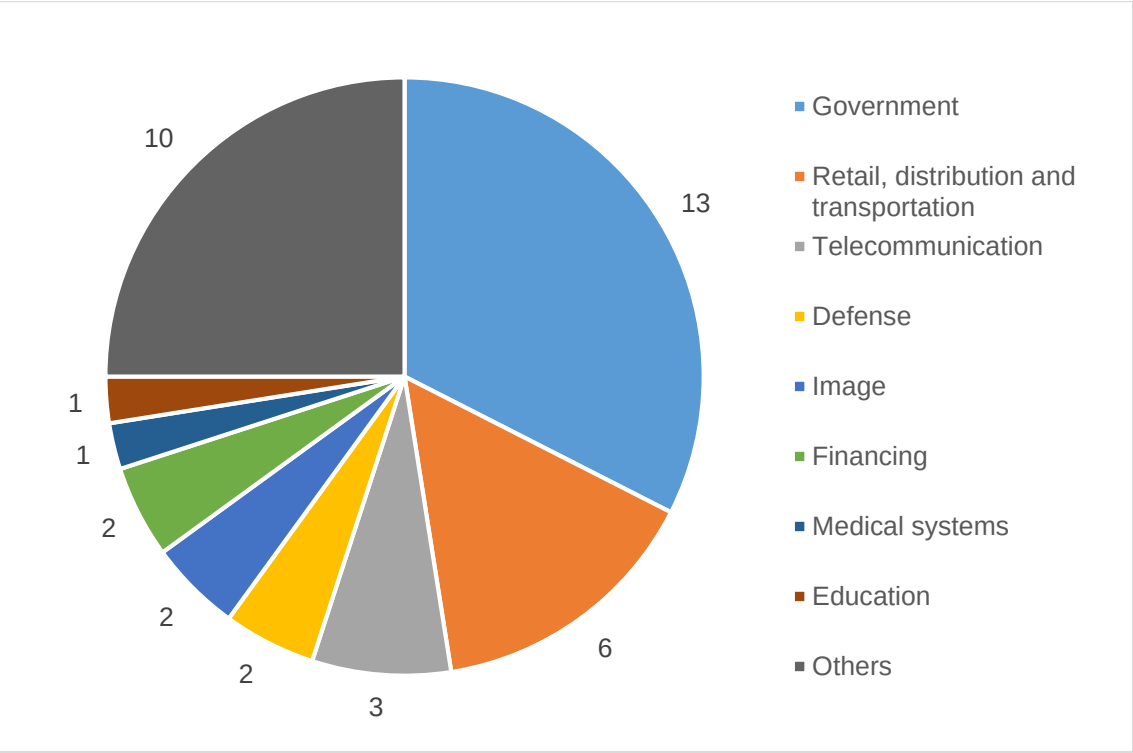

Figure 3.9 – Domain of the projects

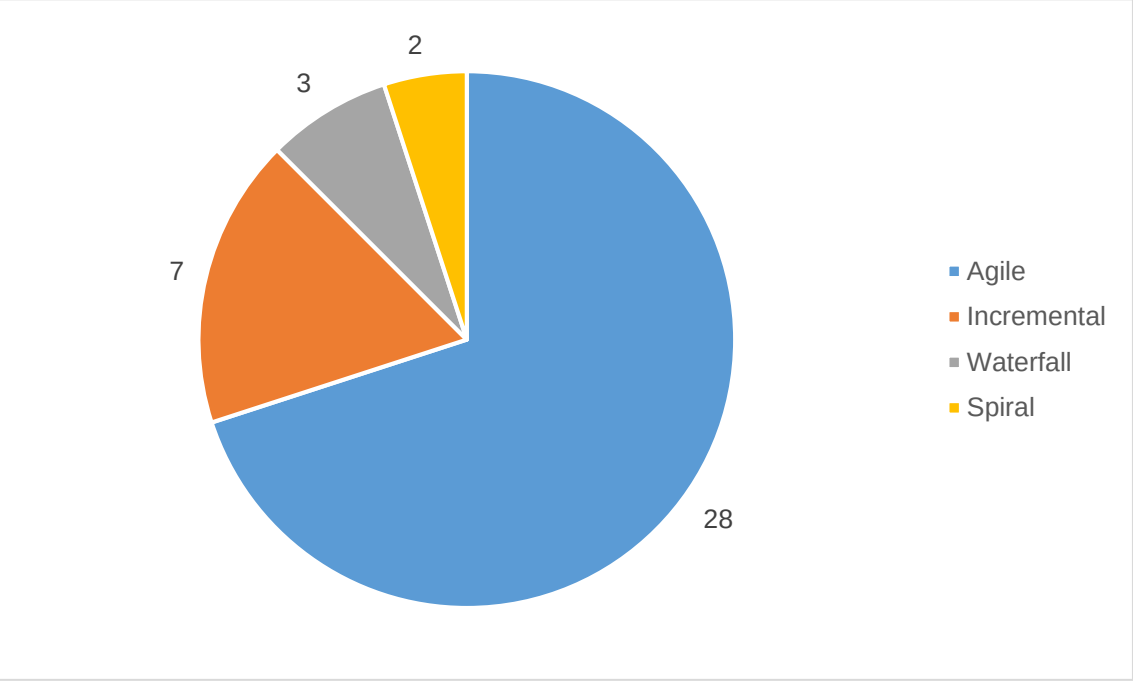

Figure 3.10 – Project development lifecycle model

3.7.2 Technical debt awareness

Regarding the perception of TD, from 40 valid answers, we observed that 16 respondents (40%) claimed not to be aware of the TD metaphor, indicating a general lack of knowledge on TD by practitioners from the industry. This result is lower than those obtained in other studies (e.g., Holvitie et al. (2018)), in which more than 80% of the participants had prior knowledge on TD, even if it was a poor one. It can indicate a poor TD dissemination on BSOs when compared to organizations from other countries.

For the 24 participants (60%) aware of its meaning, we asked to choose the options most associated with the TD definition, from which two participants did not answer. Seven issues out of twelve were selected by 50% or more of the 22 participants: low internal quality, poorly written code, that violates code rules; "shortcuts" taken during design; the presence of known defects that were not eliminated; architectural problems; issues associated with low external quality; and planned, but unfinished, tasks.

Table 3.3 – TD perception by the participants

Issue	% of participants
Low internal quality aspects, such as maintainability and reusability	77%
Poorly written code that violates code rules	68%
"Shortcuts" taken during design	68%
Presence of known defects that were not corrected	68%
Architectural problems (like modularity violation)	55%
Low external quality aspects, such as usability and efficiency	50%
Planned, but not performed, or unfinished, tasks (e.g., requirements specification, models, test plans, etc.)	50%
Trivial code that does not violate code rules	45%
Code smells	45%
Defects	36%
Lack of support processes to the project activities	23%
Required, but unimplemented, features	18%

Regarding the TD perception, from the 22 participants that were aware of the TD meaning and selected the issues related to TD, 17 informed that some issues associated with the concept of TD were perceived in their projects, while four stated that they did not perceive any issue related to TD. One participant did not answer this question.

3.7.3 Technical debt management

From the 17 participants that perceived the presence of TD in their projects, seven affirmed that neither their organization nor their project managers adopted any TDM activities, while ten stated that some TDM activities were conducted.

Out of the eight TDM activities proposed by Li et al. (LI; AVGERIOU; LIANG, 2015), as shown in Figure 3.4, only TD monitoring was not selected by any of the ten participants when asked about which TDM activities were conducted in their projects (Figure 3.11). TD identification and TD documentation are conducted in projects for six participants each, while five participants each marked TD communication, TD prioritization and TD repayment. TD measurement and TD prevention are conducted in projects for two participants. One participant did not mention any TDM activities. No participants mentioned any TDM activity besides those proposed by Li et al. (2015).

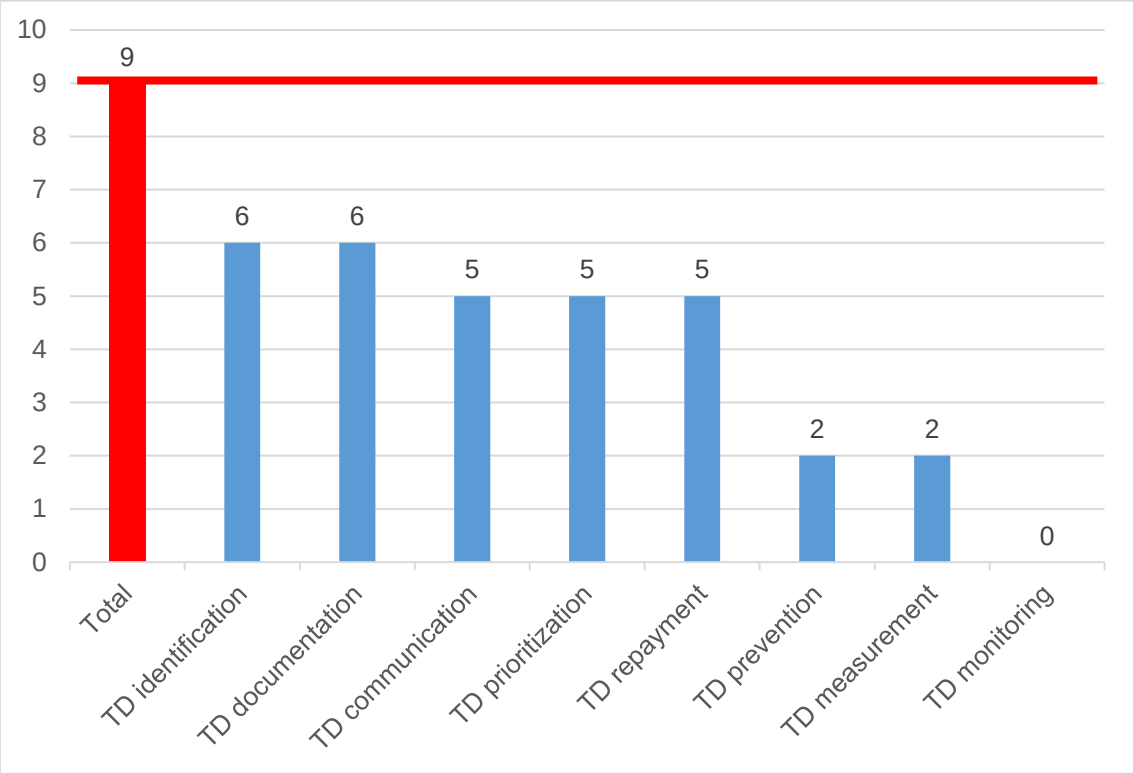

Figure 3.11 – Number of participants that conduct TDM activities

3.7.3.1 TDM responsibilities

In general, there was no consensus among participants on which roles should be responsible for each TDM activity (Figure 3.12 to Figure 3.18). The exceptions are to the development team being responsible for TD repayment, and the team leader is responsible for TD prevention.

Moreover, some conflict was observed between the participants' responses and the TDM framework, e.g., TD measurement, in which Yli-Huumo et al. (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016) state that software architects and the team leader should be responsible for the activity (see Table 3.4), whereas no respondent selected software architects as responsible for this activity.

Table 3.4 – Responsibility for each TDM activity, according to Yli-Huumo et al. (2016)

TDM activity	Responsible
TD identification	The development team, software architect
TD documentation	The development team, software architect
TD communication	The development team, software architect, team manager
TD measurement	Software architect, team manager
TD prioritization	Software architect, team manager
TD repayment	The development team, software architect
TD monitoring	Software architect, team manager
TD prevention	The development team, software architect

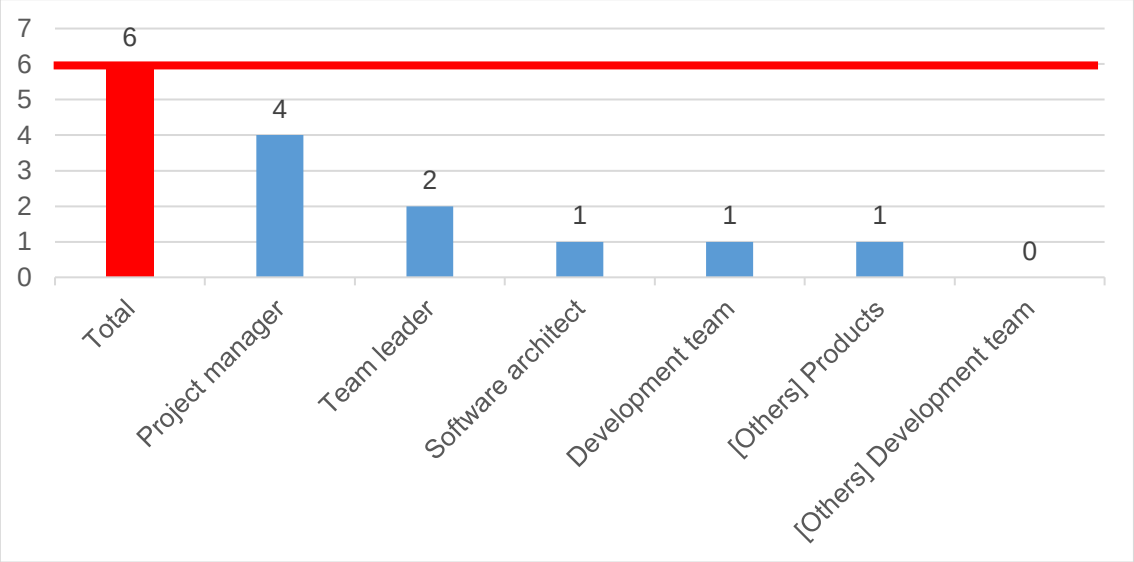

Figure 3.12 – Responsibilities for TD identification

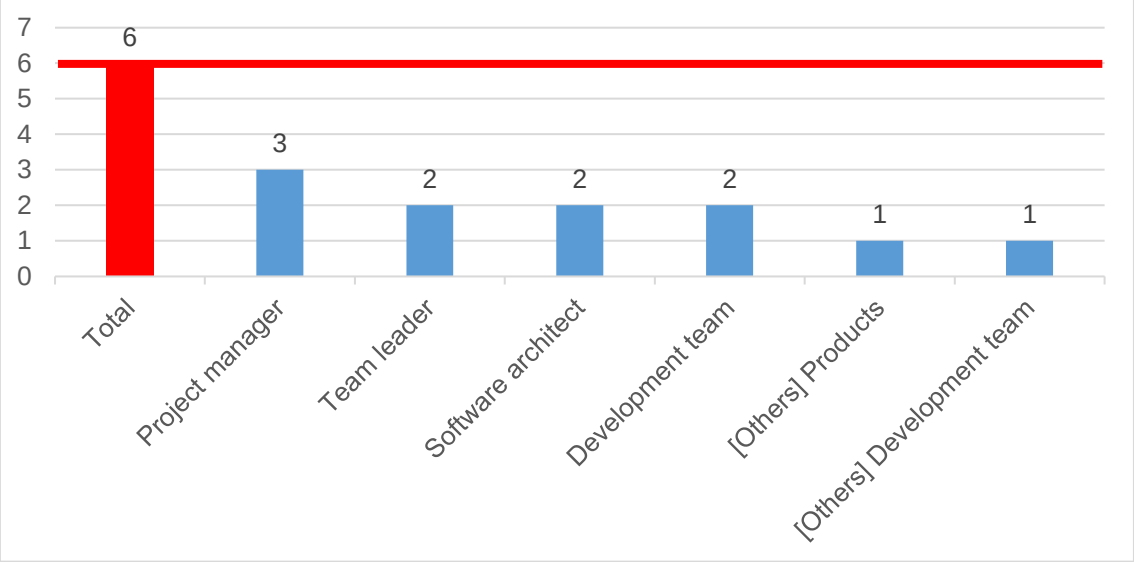

Figure 3.13 – Responsibilities for TD documentation

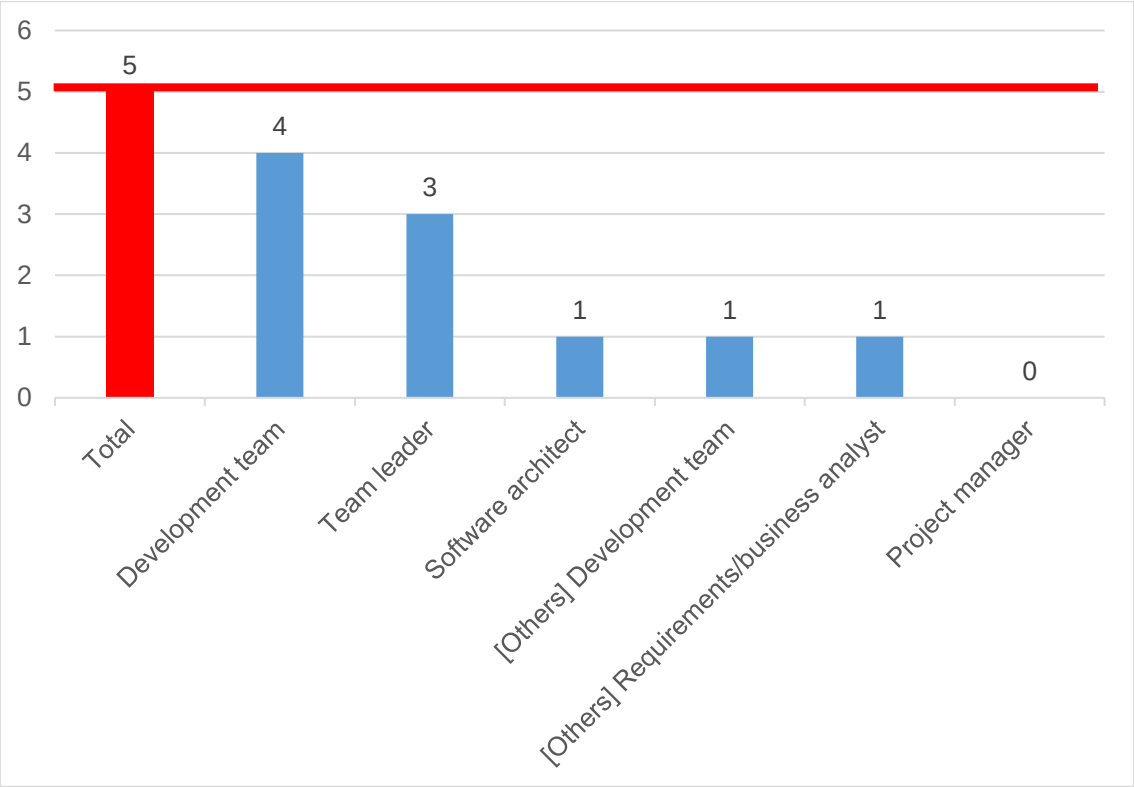

Figure 3.14 – Responsibilities for TD communication

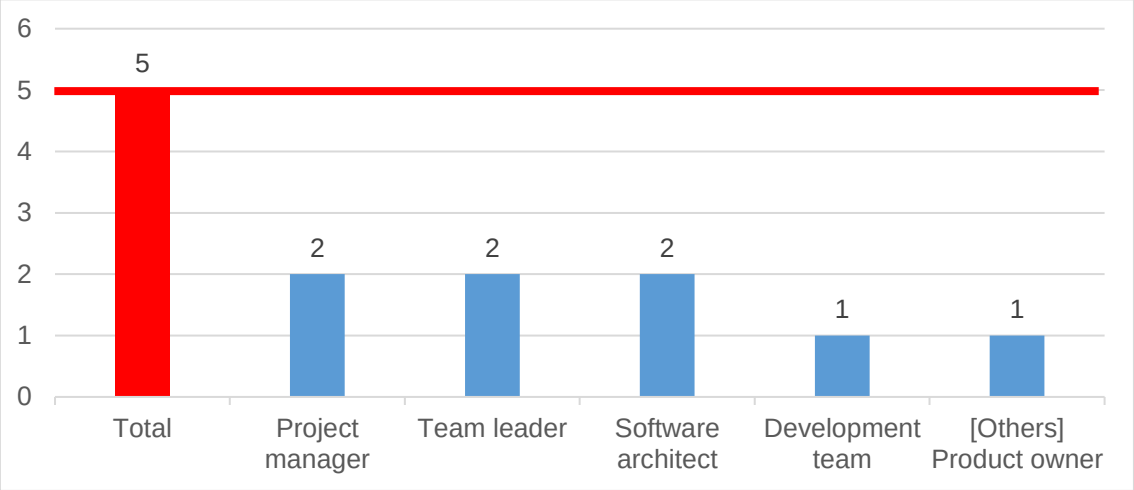

Figure 3.15 – Responsibilities for TD prioritization

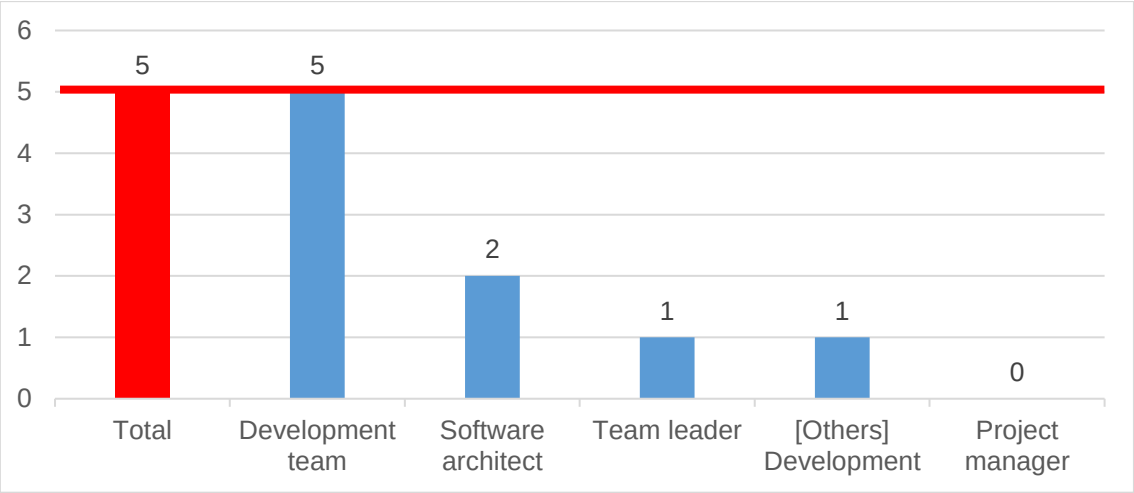

Figure 3.16 – Responsibilities for TD repayment

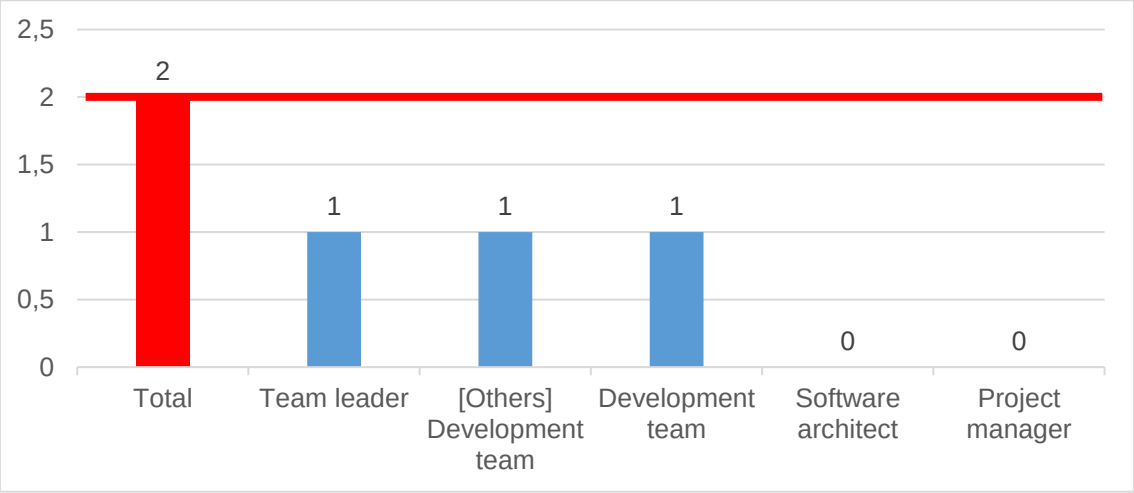

Figure 3.17 – Responsibilities for TD prevention

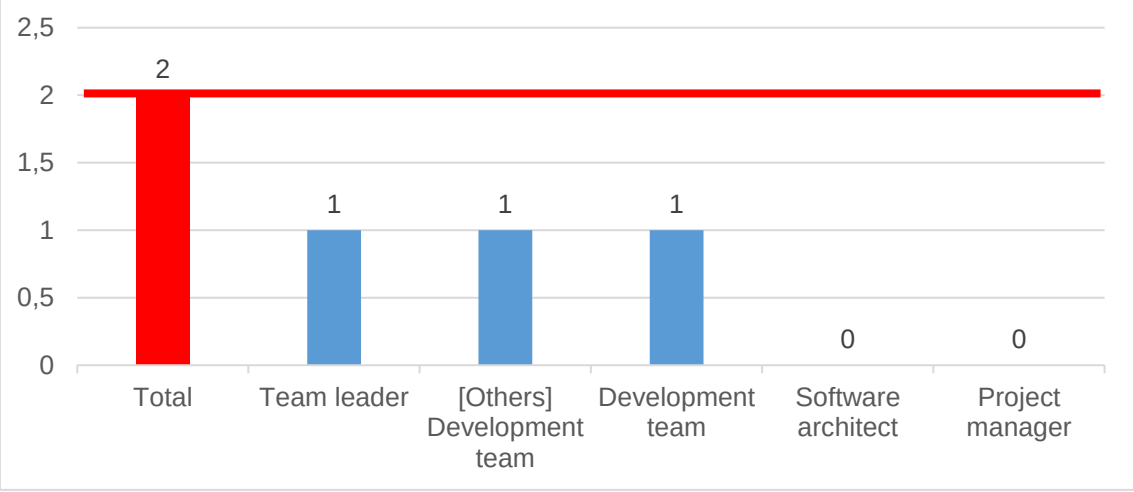

Figure 3.18 – Responsibilities for TD measurement

3.7.3.2 Most important TDM activities

Regarding the most crucial TDM activities to the nine respondents, four of them considered the TD prevention as the most relevant activity to their projects, whereas TD measurement, TD monitoring, and TD documentation were selected as least relevant by one participant each.

Table 3.5 – Evaluation of most crucial TDM activities for each participant

TDM activity	Participant id. and activity evaluation								
	20	**24**	**26**	**29**	**52**	**74**	**83**	**84**	**86**
Identification	7	5	8	7	8	5	6	6	8
Documentation	8	6	6	6	3	3	5	1	5
Communication	7	6	3	6	4	2	7	2	8
Measurement	6	6	4	5	2	1	4	3	5
Prioritization	5	8	5	7	5	8	8	5	7
Repayment	7	6	7	5	7	7	8	4	8
Monitoring	7	6	1	5	6	4	3	7	4
Prevention	8	8	2	8	4	6	6	8	4

Despite the question structure being set to enable only different grades for the TDM activities, during the release of the survey the *Javascript* code that was configured to this question was not included. Therefore, some participants repeated the same grade to different activities, e.g., participant #29 attributed grade 6 to both TD documentation and TD communication. Although this may result in a different interpretation of the results, it is still possible to obtain some answers regarding this research question, which is discussed in further details in section 3.8.

3.7.3.3 TD identification

From the total of 6 participants, two answered that there is a necessary, formal strategy to identify the TD, while the one stated that although there is a formal strategy, it is not mandatory. Three participants answered that there is no formal strategy to identify the TD.

Three participants answered that the TD identification is continuous throughout the development process, while two stated that there is no precise moment to identify TD, occurring only when a problem is observed. One participant did not answer.

Three participants answered that they have classification criteria for better organizing the TD. From these three participants, one stated that the TD is classified according to its type, considering only two possible classifications: Design Debt and Documentation Debt. Other participant informed the TD is classified according to its origin artifact. The last participant stated that it is classified considering the relation “effort to eliminate the TD vs. business value.”

3.7.3.4 TD documentation

From the total of 6 participants, two answered that they have a standard on documenting the TD that should be followed by all stakeholders. One participant answered that his/her project has a TD documentation standard, but it is not mandatory for the stakeholders. Two participants answered that the TD documentation is conducted only informally.

When asked how the TD is documented, four participants answered that they use a general task backlog, with no specific details, while one affirmed he/she uses a specific backlog of TD items.

One participant did not provide any details on TD documentation, despite informing that it is conducted in his/her project.

3.7.3.5 TD communication

From the five participants that answered the TD communication section, four affirmed that the TD was discussed during project meetings, but with the participation of only a few of the necessary stakeholders. One participant said that the TD was only discussed informally.

3.7.3.6 TD measurement

Out of the two participants that answered the TD measurement section, one affirmed that the activity was conducted informally, through the analysis of metrics and indicators based on specific information regarding the TD item. The other participant indicated that there is a formal strategy to measure TD, based on pure information, like the number of TD items.

When asked which information or variables are used to measure the TD, one participant affirmed using the relation “estimated effort vs. value,” where the other answered using persons-hour or persons-month and LOC of the TD item.

3.7.3.7 TD prioritization

Regarding the TD prioritization, three of the five participants answered that the TD items were prioritized according to “guesses” or estimates based on previous experiences, while one participant used the criticality of the TD item to prioritize it.

Four participants affirmed that they tend to prioritize the TD items that most impact the client, and three participants answered that they prioritize the TD items that could cause most impact the project. Two participants adopt the criterion of TD items that have the highest level of severity. The criteria “debt items located in widely used parts of the system” and “debt items that have a bad cost/benefit relation” were selected by one participant each.

One participant did not provide any details on TD prioritization, despite informing that it is conducted in his/her project.

3.7.3.8 TD repayment

From the five participants that answered the TD repayment section, two answered that the TD repayment is planned according to the current necessities of the project, while one participant answered that the TD repayment is planned continuously, with specific periods during the development process destined to this activity. One participant answered that the TD is only repaid when there is not possible to avoid it anymore.

One participant did not provide any details on TD repayment, despite informing that it is conducted in his/her project.

3.7.3.9 TD prevention

Both respondents of this section answered that TD prevention is an activity conducted informally, by each member of the team individually.

3.7.3.10 Technologies for TDM

Table 3.6 presents a list of practices, techniques, and tools used in each TDM activity. The numbers in parentheses represent the number of participants answering that specific section (column “TDM activity”) and the number of participants that affirmed using that tool or technique (column “Tools and techniques”). We can observe that different technologies support TDM and there is no consensus about which one to use.

3.8 Answering the research questions

3.8.1 Answering RQ1

We did not observe consensus in any available option regarding the perception of TD. Twelve issues about TD were presented in the questionnaire, and each participant was asked to select which of them should be associated with the TD concept, as presented in Table 3.3. Out of those options, only one issue was evaluated by 75% or more of the 22 respondents. From the issues associated with the TD concept by 50% or more of participants, only one (“issues associated with low external quality” and “planned, but unfinished, tasks”) are not considered TD, as per Li et al. (2015). However, as discussed in chapter 2, “planned, but unfinished tasks” must be considered as TD according to the *Dagstuhl* definition (AVGERIOU et al., 2016).

Table 3.6 – TDM activities – tools and techniques

TDM activity	Tools and techniques
TD identification (6)	Manual code inspection (4), dependency analysis (1), checklist (2), *SonarQube*[3]/*SQALE*[4] (3), *CheckStyle*[5] (1), *FindBugs*[6] (1)
TD documentation /representation (6)	TD backlog (3), specific artifacts for TD documentation (1), *JIRA*[7] (1), others - *Trello*[8] (1)
TD communication (5)	Discussion forums (3), specific meetings about TD (1), others - *GitLab*[9] (1), others – *Trello* (1)
TD measurement (2)	Manual measurement (1), *SonarQube* (2), *JIRA* (1)
TD prioritization (5)	Cost/benefit analysis (1), classification of issues (3)
TD repayment (5)	Refactoring (3), redesign (1), code rewriting (4), meetings/workshops/training (1)
TD monitoring (0)	N/A
TD prevention (2)	Guidelines (2), coding standards (2), code revisions (1), retrospective meetings (1), Definition of Done (2)

Only one issue associated with TD concept was marked by less than 50% of the 20 answers, which is “Code smells” (45%). Therefore, we believe that there is some small agreement among participants of what should be considered TD since 17 out of 22 participants identified that TD should be related to internal quality issues.

However, 50% of the participants believe that TD should also be associated with external quality issues. It is worrisome and contradicts the definition asserted at the *Dagstuhl* Seminar (AVGERIOU et al., 2016). It could indicate that there is a misconception of what should be considered TD, associating its definition with any issue occurring during the software development, and possibly overusing the term.

We could observe some alignments in the views on TD between the participants and academia since most of the issues associated by more than half of the participants are also in agreement with the definition indicated in the technical literature. We believe that despite the reasonable TD definition understanding by some software practitioners, it is vital to disseminate better the distinction between issues related to internal quality (TD) and those related to external quality (defects).

[3] https://www.sonarqube.org

[4] http://www.sqale.org/

[5] http://checkstyle.sourceforge.net/

[6] http://findbugs.sourceforge.net/

[7] https://br.atlassian.com/software/jira

[8] https://trello.com

[9] https://about.gitlab.com/

3.8.2 Answering RQ2

Only 43% of 40 participants claimed to perceive TD in their software projects, which could be considered low, considering the importance of the topic. Moreover, the number of participants mentioning they include any TDM activities in their projects is 25%, indicating the possible existence of a severe gap in the product overall quality perspective.

However, the fact that there is a low perception of the TD presence in the surveyed organizations does not eliminate the possibility of the adoption of other internal quality assurance methods.

3.8.3 Answering RQ2.1.1

There was no consensus on which TDM activities are more relevant to software projects. However, almost half of the participants that answered this question (four out of nine) mentioned that TD prevention is relevant to a project. However, only 20% of the participants that answered which TDM activities they conduct affirmed adopting any TD prevention tasks. It can indicate a possible bias from the participants' point-of-view, as they could believe that the "correct" answer to "which TDM activity is most relevant" was TD prevention.

Regarding the most conducted TDM activities by the participants, our results are mostly in line with Yli-Huumo et al. (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016) in which it is indicated that TD communication is most commonly adopted by the development teams, followed by TD identification, documentation, prioritization, repayment, and prevention. The rarely managed TD activities described in (YLI-HUUMO; MAGLYAS; SMOLANDER, 2016) are TD measurement and TD monitoring, as also observed in our study.

3.8.4 Answering RQ2.1.2

As presented in section 3.7.3.10, a list of tools and technologies used to manage TD activities (see Table 3.6) can be used in further studies looking for evidence on their effectiveness and efficiency in managing the TD.

3.9 Threats to validity

The main threat to validity is the generalization of results. Since the target sampling is non-probabilistic, it is not possible to determine *a priori* the population size and the expected total number of participants. Then, the results confidence level might be low, making hard to generalize the results to the entire population (BSOs). Despite that,

the inevitable conclusions can suggest the TD research with initial indications of the level of knowledge of BSOs regarding the TD concept and TDM activities.

A potential internal threat comes from the participants that might have misunderstood some terms and concepts of the questionnaire based on different experiences.

There is also a construct threat of a biased survey, from the researchers' perspectives and the collected information from the technical literature such as the TDM activities organized in Li et al. (2015). To reduce the level of this menace, we conducted a series of revision cycles during the survey development with two researchers. Furthermore, two pilot trials were executed, followed by a final revision by all the pilot survey participants aiming to ensure the modifications were aligned with their perspectives.

3.10 Chapter considerations

This chapter presented the execution of a survey conducted by practitioners in BSOs. The results provide initial observations regarding how BSOs (represented by their software practitioners) perceive and manage TD in their projects.

Before the analysis of the survey results, some observations can be made. First, we obtained a considerable low response rate and an even lower complete response rate. Only 65% of the 62 responses were valid to the point we could obtain some information. It could be caused mainly by the survey length. Overall, the survey has 52 questions (no participant had to answer the complete survey, though), distributed over 23 pages. Studies report that every additional question can reduce the response rate by 0.5%, and every additional page, by 5% (LINÂKER et al., 2015). Notwithstanding, the results were enough to provide an initial and representative picture of the TDM scenario in BSOs.

Another possible reason for a low response rate is that the concept of TD is still incipient in BSOs and, since the topic was explicitly mentioned in the invitations, it could have kept away some practitioners that are not familiar with the term. If this case is indeed real, the results would be even more worrisome, as the percentage of practitioners that know what TD is could drastically drop. However, since we do not have data on this possibility, we cannot formulate any elaborate conclusions.

Regarding the TD awareness, a low percentage of practitioners that know the concept of TD indicates the lack of technologies or information on TD available to the industry. It is curious, though, as the definition itself originated in the industry. In any way, the results indicate a possible research gap in developing better ways to export the current knowledge on TD developed by the academia to the industry.

Regarding the TD understanding, our results indicate no unanimity concerning on how Brazilian software practitioners perceive TD. With the results from the TD awareness, we can observe a broader issue in the context of BSOs: most of the practitioners might not know what TD is, and the ones who know do not fully understand what it indeed is. Developing ways to disseminate the current definition of TD to the industry is essential to provide better technologies on this matter.

Regarding TDM, it was observed that only a few BSOs report the TD management in their software projects, indicating that TDM seems to be still incipient in BSOs. Four out of nine practitioners reporting TDM activities claimed that TD prevention is the most critical activity in their projects, despite only two participants indicated to perform it. From our study, it is not possible to understand why only some organizations adopt TDM activities. Based on our other findings, we could suppose it is because of the small number of practitioners that know what TD is, but there could be several other reasons, from adding more processes to the possible lack of user-friendly technologies available in the Brazilian market.

Overall, we believe that this study offers a novel perspective on the TD research in BSOs. Based on the previously conducted literature review, this survey differs from other studies of the kind as it provides a general overview of the current perspective on TD and TDM in BSOs. To the best of our knowledge, only one other survey analyzed the TD specifically in BSOs (ROCHA; ZAPALOWSKI; NUNES, 2017). Despite studying the same context, their research focuses mainly on the TD located at the code level, and not on a broader point-of-view like ours.

The results of this survey provide the following contributions to both industry and academia:

- To the BSOs (industry) - the initial results indicate that software practitioners and their organizations need to understand the concept of TD better to achieve better results in their projects, as the perception of TD and its management in this scenario is still incipient. The findings also present a list of technologies that can be used to support TDM activities, as long as the software engineers evaluate their usage based on the organizations' necessities at the time. Moreover, the findings indicate that the TDM activities usually involve distinct roles throughout the projects;
- To the researchers - our results indicate that there is a need to more investigations aiming to disseminate the TD knowledge to practitioners on BSOs, as well as to provide strategies and software technologies to support the TDM in such organizations. Besides, we believe the sharing of this study package can contribute to support the development of in-

vestigations on TD and its management more connected to the software organization needs in Brazil and other regions.

4 Technical debt management technologies

This chapter presents a quasi-Systematic Literature Review conducted to assess which new technologies were developed and reported in the technical literature, aiming to manage the technical debt. The results obtained in this review point to different technologies that are already available to the industry but are not yet adopted by Brazilian practitioners.

4.1 Introduction

As discussed in the previous chapter, we could observe that TDM is still incipient in BSOs. It is due to several reasons, such as the lack of understanding of the TD concept, its scope, limitations, the lack of technologies to support TDM or the lack of knowledge of existing technologies.

Some studies have reported on existing technologies being adopted in organizations, such as Yli-Huumo et al. (2016; 2016), Oliveira et al. (2015), Assunção et al. (2015), Ernst et al. (2015), and Zazworka et al. (2013). Other papers report evaluations of the most common technologies already commercially available, such as *CLIO*, *Codevizard,* and *Findbugs* (ZAZWORKA et al., 2014), SQALE and *SonarQube* (GUAMAN et al., 2017a). Finally, some software practitioners from the industry reported their practices and adopted technologies for TDM in the software development, such as dos Santos et al. (2013), Krishna (2013) and Eisenberg (2012).

However, to provide a proper contribution to the industry, it is also essential to assess which new software technologies were created by the academic community and were not yet communicated to the industry. This chapter presents the results of a secondary study conducted to reveal the gap regarding this topic between the industry and the academic community. This study was undertaken by this author and another researcher from the Experimental Software Engineering Group (ESE) at COPPE-UFRJ.

First, it is presented an analysis of the related works, including primary studies comparing or analyzing the current technologies. Following that, the concepts of a literature review and its planning are presented. Finally, the results are discussed, along with the threats to validity and some final considerations of this research.

4.2 Related works

4.2.1 Primary studies on TDM technologies

Through the study conducted by Yli-Huumo et al. (2016) (discussed in section 2.6.3), a list of tools and practices associated with each TDM activity has been organized. Those can be associated with a group of individual practices, as presented below:

- TD repayment: refactoring, redesigning, rewriting;
- TD prevention: coding standards, code reviews, Definition of Done
- TD representation/documentation: TD backlog/list, documentation practice, project management tools (*JIRA*, *Wiki*);
- TD identification: time reservation for manual code inspection, use of code analysis tools (*SonarQube*, *Checkstyle*, *FindBugs*);
- TD measurement: data from measurement tools (*SonarQube*) and data from project management tools (*JIRA*, *Wiki*);
- TD monitoring: monitoring tools (*SonarQube*), project management tools (*JIRA*, *Wiki*);
- TD communication: specific TD meetings, TD included in discussion topics;
- TD prioritization: cost/benefit model, issue rating.

Oliveira et al. (2015) applied the TD template introduced by Seaman and Guo (2011) on action research based studies with software development teams adopting agile methodologies. They also reported the use of the *Vtiger* and *Trello* tools to document TD items, through a TD backlog. Another team used the *Jira* tool to document TD items, including them on a general product backlog.

Zazworka et al. (2013) executed a study to evaluate how a software team identifies TD. The practitioners conducted two identification activities: manual identification, using Seaman and Guo's (2011) TD template, and automatic identification, applying the *Codevizard* and *FindBugs* tools to the latest version of the subject project source code. One of their conclusions was that many TD items could not be identified through the adopted tools, making necessary to conduct the TD identification manually, in parallel to the automatic identification, and/or researching for automatic tools to identify TD in other artifacts besides the source code.

Yli-Huumo et al. (2016) conducted an action survey to develop new processes to manage TD in a Scandinavian organization. They adopted a similar TD identification template as described in Seaman and Guo (2011), alongside with an evaluation template to be filled for all identified TD items.

Table 4.1 – Template for TD documentation (adapted from (YLI-HUUMO et al., 2016))

Technical Debt ID	Technical debt identification number
Date / Reporter	Reporting date / Reporter name
Technical debt name	Name of the identified technical debt
Description	Description of the identified technical debt
Alternatives	Explanation of possible alternative solutions
Rationale	Reasons to fix the technical debt

Table 4.2 – TD evaluation template (adapted from (YLI-HUUMO et al., 2016))

#	**Question**
1	What are the benefits of fixing this issue? (Business value, quality, productivity, fewer defects, etc.)
2	Are there any risks in fixing this issue? (It is expensive, breaks the system, etc.)
3	Why as this issue done previously like that?
4	How to fix this issue and what resources the fix would require?
5	From scale 1 – 5, how important would you rank this issue to be dealt? (1 – most important, 5 – not so important)

Assunção et at (2015) reported the application of a model to identify and evaluate code-related TD items. They adopted the SQALE quality model (2016) to identify the TD and calculated the principal of the TD items through the formula proposed by Curtis et al. (2012), with some adaptations due to the project's domain. The adapted formula is:

$$TD\ Principal = ((VCHR)\ x\ (Avg.Hours\ to\ fix)\ x\ (\$\ per\ hour) + (VCMR)\ x\ (Avg.Hours\ to\ fix)\ x\ (\$\ per\ hour) + (VCLR)\ x\ (Avg.Hours\ to\ fix)\ x\ (\$\ per\ hour)$$

Where:

- VCHR is the number of violations at high risk to be corrected;
- VCMR is the number of violations at medium risk to be corrected;
- VCLR is the number of violations at low risk to be corrected.

The number of violations at each risk level was defined according to the expression:

$$VCr = VTr - (LOCS * DRr)$$

Where VCr is the number of violations to be corrected at that level, VTr is the total number of violations, LOCS is the number of lines of code and DRr is the density of violations acceptable for code line, for each risk level.

Ernst et al. (2015) conducted a survey, as described in section 3.2.1. Among their research questions, they investigated which practices and tools were adopted to manage the TD. They found that specific TDM tools were rarely used, even though some of them were installed. The main reason to not use them was the complexity of configuring the software and interpreting the results. 50% of the participants stated that they did not use any tool to manage the TD. Through interviews, the participants answered that tools could produce many false positives, and often require specific customization. The participants using tools have listed *Redmine*, *Jira*, *Team Foundation Server*, *SonarQube*, *Understand*, *CPPCheck*, *FindBugs,* and *Sloccount*.

Tonin (2018) reported a series of studies in nine academic teams at a Brazilian university and in five teams at BSOs, aiming to identify what TD types are incurred by practitioners in agile software development teams, to assess the impact on teams of TD in their projects, and to list techniques and tools used by these teams to identify, monitor and manage the TD, among other objectives.

Her findings include an observation that test debt, architecture debt, and documentation debt were the TD types most frequent in the cases studied. Regarding the impacts of TD on the agile software development teams, she reported the direct impact on business, impairment on team motivation and an elevation on the general stress level of the teams. Finally, she reported a set of technologies used to manage TD, including the inclusion of a specific TD section on the Scrum board, the use of the *Jira* tool, the tools *SonarQube* and *PMD* to monitor the TD, a *Sonar* plugin on the *Jenkins* server to identify the TD on a daily basis, and other code quality analysis tools, such as *Checkstyle*, *Coverage* and *PMD*.

4.2.2 Experience reports from industry's practitioners

Dos Santos et al. (2013) reported the use of a TD board (see Figure 4.1) to document and communicate the TD items between team members. Among the lessons learned, they claimed the necessity to make visible the TD items, encouraging the teams to self-evaluate their debts. They also observed that the communication was improved through the clear visualization of TD among the teams, as well as a faster repayment rate.

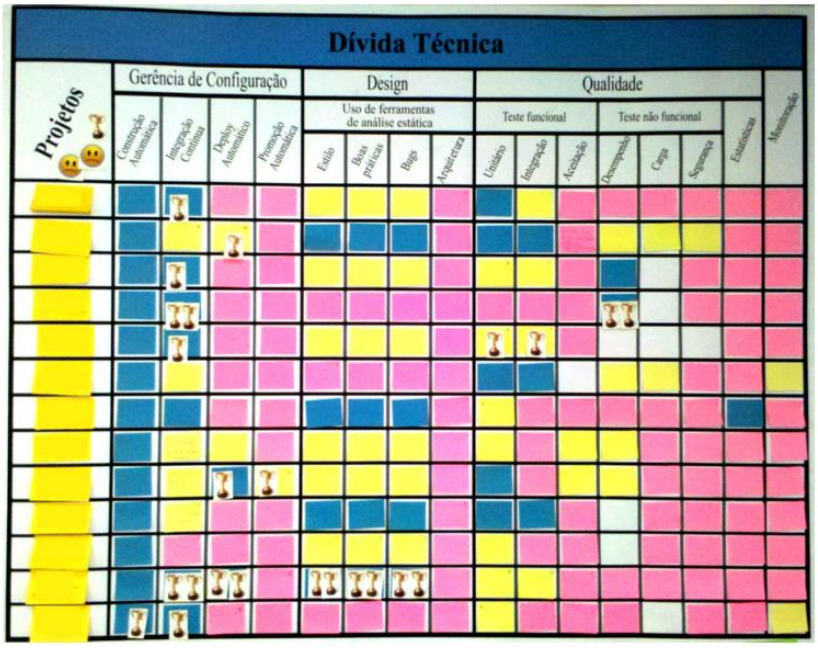

Figure 4.1 – Representation of a TD board (reproduced from dos Santos et al. (2013))

Krishna and Basu (2012) classified the TDM practices into three groups: identification, classification, and reduction. Despite being anecdotal, the authors provide a sequence of steps for reducing TD, while it is possible to mention:

- Dedicate daily/weekly time to review and refactoring code;
- “Smell one’s own code,” or acknowledge the presence of wrong or unwanted code, attempting to reduce/remove it whenever possible;
- Adopt best practices and code standards;
- Learn and apply refactoring techniques;

Eisenberg (2012) introduced an approach to managing the TD, based on four steps:

- Use available tools to analyze the code, both statically and dynamically;
- Use known data to evaluate the cost to repay the debt;
- Calculate the total time to repay the debt;
- Use the cost data to determine the monetary cost.

To analyze the code the author adopted *SonarQube*, despite not using the debt calculator within the tool, as the measurement strategies were not coherent to his company’s context. However, other metrics obtained from the tool were used, such as duplicate code, method or class complexity, automated and manual test coverage and package interdependencies.

Following the metrics, Eisenberg guides prioritizing the TD items, based on development methodology, failure criticality, security requirements, lifecycle phase,

planned reuse, requirements and project status. After that, an example of thresholds is presented. For each metric, three statuses are defined (green, yellow and red), with percentages in which the metric should be monitored. A metric with green status is manageable, whereas the yellow and red statuses should be avoided.

Table 4.3 – TD thresholds (adapted from (EISENBERG, 2012))

		Status		
		Green	Yellow	Red
Duplicate code	<=	5%	10%	otherwise
Rules compliance	>=	80%	60%	otherwise
Comments - Interface	>=	90%	75%	otherwise
Comments - General	>=	30%	20%	otherwise
Package interdependencies	<=	5%	10%	otherwise
Method complexity > 8	<=	10%	15%	otherwise
File/class complexity > 60	<=	10%	15%	otherwise
Test coverage - Automated	>=	80%	60%	otherwise
Test coverage - manual	>=	80%	60%	otherwise

With the thresholds established and the measurement tools configured, there is the need to define the costs associated with the repayment of each type of debt, i.e., removing a duplicated code block or removing a code violation. The author presents reference values, which should be tailored to each project's context. Finally, with all information in hands, the author presents the implementation of a debt calculator, using Microsoft Excel.

Table 4.4 – Remediation costs (adapted from (EISENBERG, 2012))

Occurrence	**Hours**
Remove duplicate code block	2,0
Automate uncovered line/branch	0,2
Manually test uncovered line/branch	0,4
Remove rule violation	0,1
Add interface commentary	0,2
Add comment line	0,1
Remove package interdependency	4,0
Update/split complex method	0,5
Update/split complex file/class	8,0
Cost per labor hour	$ 60,00

Table 4.5 – Summary of the threshold debt calculator (adapted from (EISENBERG, 2012))

	Month 1	Month 2	Month 3
Summary level Info			
SLOC	60.582	72.306	83.180
Total debt (hours)	3.746	3.352	2.597
Total debt $	$ 224.731,00	$ 201.143,00	$ 155.830,00
Debt/KSLOC	$ 3.710,00	$ 2.782,00	$ 1.873,00
Debt breakdown			
Duplicate code	13%	0%	0%
Test coverage - automated	62%	86%	87%
Rules compliance	7%	3%	0%
Code comments	17%	10%	13%
Package interdependencies	1%	0%	0%
Complexity	0%	0%	1%

4.2.3 Secondary studies on TDM technologies

Ribeiro et al. (2016) conducted a mapping study to investigate the decision criteria most commonly adopted to repay TD items. They came out with a set of criteria, divided into four categories: nature of the TD, customer, effort, and project. A follow-up study (RIBEIRO et al., 2017) consolidated a strategy to prioritize TD items based on those criteria. For each criterion, a yes or no question is formulated. The total number of "yes" answers (weighted, if necessary to the context) offers a ranking of TD items. Those with higher scores should be prioritized during the repayment. The questions are presented in Table 4.6.

Table 4.6 –TD prioritization based on decision criteria (adapted from Ribeiro et al. (2016))

Category	Criteria	Question
Customer	Analysis when the refactored part will be used	Are the system features affected by debt used frequently?
Customer	Debt impact on customer	Does this debt directly impact the use of the system?
Customer	Visibility	Can the user perceive the debt effect?
Effort	Debt impact on the project	Is the amount of debt-affected components large?
Effort	Cost-benefit	This debt has a good cost/benefit?
Effort	The effort to implement the proposed correction	Is the effort to pay off this debt small?
Nature	The severity of the debt	Does this debt have high severity?
Nature	Localization of TD	Are there current development and/or maintenance activities occurring in parts of the project where the debt is located?
Project	Nature of the project	Is this project critical?

Li et al. (2015) conducted a mapping study to gather, among other information, TDM activities, TD types under-studied by researchers, notions used for describing and explaining TD, quality attributes compromised when TD is incurred and tools and practices supporting TDM activities. The practices obtained through the mapping study are presented in Table 4.7.

Table 4.7 – Practices to support the TDM activities (adapted from Li et al. (2015))

TDM activity	Technologies	Description
TD identification	Code analysis	Analyze source code to identify violations of coding rules, lack of tests and design or architecture issues
	Dependency analysis	Analyze dependencies between different types of software elements
	Checklist	Check against a list of predefined scenarios where TD is incurred
	Solution comparison	Compare the actual solution with the optimal solution on some dimension
TD measurement	Calculation model	Calculate TD through mathematical formulas or models
	Code metrics	Calculate TD using metrics of source code
	Human estimation	Estimate TD according to experience and expertise
	Cost categorization	Estimate various types of the cost of handling the incurred TD
	Operational metrics	Indicate TD using quality metrics of product operation
	Solution comparison	Calculate the distance between the actual solution and the optimal solution
TD prioritization	Cost/benefit analysis	If resolving a TD item can yield a higher benefit than cost, then this TD item should be repaid. TD items with higher cost/benefit ratios of repayment should be repaid first
	High remediation cost first	TD items that cost more to resolve should be repaid first
	Portfolio approach	The portfolio approach considers TD items along with other new functionalities and bugs as risks and investment opportunities (i.e., assets)
	High interest first	TD items incurring higher interest should be repaid first
TD monitoring	Threshold-based approach	Define thresholds for TD related quality metrics, and issue warnings if the thresholds are not met
	TD propagation tracking	Track the influences of TD through dependencies between other parts of a system and the parts of the system that contains TD
	Planned check	Regularly measure identified TD and track the change of the TD
	TD monitoring with quality attribute focus	Monitor the change of quality attributes that detrimental to TD, such as stability
	TD plot	Plot various aggregated measures of TD over time and look at the shape of the curve to observe the trends

TDM activity	Technologies	Description
TD repayment	Refactoring	Make changes to the code, design, or architecture of a software system without altering the external behaviors of the software system, to improve the internal quality
	Rewriting	Rewrite the code that contains TD
	Automation	Automate manually-repeated work, e.g., manual tests, manual build, and manual deployment
	Reengineering	Evolve existing software to exhibit new behaviors, features, and operational quality
	Repackaging	Group cohesive modules with manageable dependencies to simplify the code
	Bug fixing	Resolve known bugs
	Fault tolerance	Strategically place runtime exceptions where the TD is
TD communication	TD dashboard	A dashboard displays TD items, types, and amounts to get all stakeholders informed of the existence of the TD
	Backlog	All identified TD items, as well as anything to be resolved in the development, are put into the backlog of the software project
	Dependency visualization	Visualize the undesirable dependencies between software elements
	Code metrics visualization	Visualize code metrics in some tools such as code maps and highlight s software elements with bad measured quality
	TD list	A TD list keeps all identified TD items and makes them visible to stakeholders
	TD propagation visualization	Show the connections between different TD items, and how a TD item affects and is affected by other TD items
TD prevention	Development process improvement	Improve current development processes to prevent the occurrences of certain types of TD
	Architecture decision making support	Evaluate potential TD caused by different architecture design options, and then choose the option with less potential TD
	Lifecycle cost planning	Develop cost-effective plans that look at the system throughout the lifecycle to minimize overall TD of the system
	Human factors analysis	Cultivate a culture that minimizes the unintentional TD caused by human factors, e.g., indifference and ignorance

In particular, for TD documentation, a list of fields to be completed by the developer or the responsibility for that activity is indicated in Table 4.8.

Table 4.8 – Fields of a TD item documentation (adapted from Li et al. (2015))

Field	Description
ID	A unique identifier for a TD item
Location	The location of the identified TD item
Responsible/author	The person who is responsible for repaying the TD item
Type	The TD type that this TD item is classified into, e.g., architectural TD
Description	General information on the TD item
Date/time	The date or time when the TD item is identified
Principal	The estimated cost of repaying the TD item
Interest amount	The estimated extra cost of tolerating the TD item
Interest probability	The probability that the interest of this TD item needs to be repaid
Interest standard deviation	The estimated difference between the estimated interest amount and the actual (future) interest amount
Correlations with other debt items	Relationships between this TD item and other TD items
Name	The name of a specific type of TD in a TD item (in case a TD has multiple TD types)
Context	A certain implementation context (e.g., the programming language used) of a specific TD type in a TD item
Propagation rules	How this TD item impacts the related parts of the software system
Intentionality	Intentionally or unintentionally incurred

Finally, a list of tools was created, indicating their functionality, vendor, TD types and artifacts covered. This list is summarized in Table 4.9. In this table, tools marked with an asterisk (*) were explicitly built for TDM.

Table 4.9 – Tools to support TDM activities (adapted from Li et al. (2015))

TDM activity	Tool
Identification	SIG Software Analysis Toolkit, Rational AppScan, PMD, PHPMD, NDepend, NCover, FxCop, CodeXpert, Cobertura, Checkstyle, Software maps tool*, RE-KOMBINE, CAST's Software Applications Intelligence Platform, Technical Debt Evaluation (SQALE) plugin for SonarQube*, Resource Standard Metrics, DebtFlag*, RBML compliance checker, A tool to identify adverse dependencies, Sonar TD plugin*, SonarQube, SonarQube COBOL plugin, CLIO, CodeVizard, FindBugs
Measurement	Sig Software Analysis Toolkit, Google CodePro Analytix, iPlasma, Eclipse Metrics, Software maps tool*, RE-KOMBINE, Technical Debt Evaluation (SQALE) plugin for SonarQube*, STAN*, Resource Standard Metrics, Sonar TD plugin*
Prioritization	Technical Debt Evaluation (SQALE) plugin for SonarQube*, Sonar TD plugin*
Monitoring	Software maps tool*, DebtFlag*, Sonar TD plugin*
Repayment	Sonar TD plugin*
Documentation	DebtFlag*
Communication	Software maps tool*, Code Christmas Trees, Technical Debt Evaluation (SQALE) plugin for SonarQube*, DebtFlag*, Sonar TD plugin*
Prevention	N/A

From the tables above, it is possible to observe that only a few tools revealed by the mapping study were developed to manage TDM: Software maps tool, SQALE plugin for *SonarQube*, *DebtFlag*, *Sonar* TD plugin and STAN. This information, along with evidence that these tools are rarely used by software practitioners (ERNST et al., 2015), we observed a research gap on technologies explicitly developed to manage the TD, which serves as the primary motivation to conduct this literature review.

4.3 Systematic literature reviews

A systematic literature review (SLR), or just systematic review, is a form of secondary study, i.e., it intends to review primary studies concerning a set of research questions, with the primary goal being to synthesize evidence and/or knowledge regarding the set of research questions.

The literature review is considered systematic when it takes into consideration a defined methodology to conduct the search, analysis and data extraction from the primary studies found in the review. Two main advantages to conducting an SLR instead an *ad hoc* literature review is the reduced bias from the researchers and the more straightforward replicable process, simplifying the validation process from other researchers and facilitation future reviews on the protocol (KITCHENHAM, 2007).

According to Biolchini et al. (2007), the process to conduct an SLR consists of four phases, as indicated in Figure 4.2, and detailed from now on.

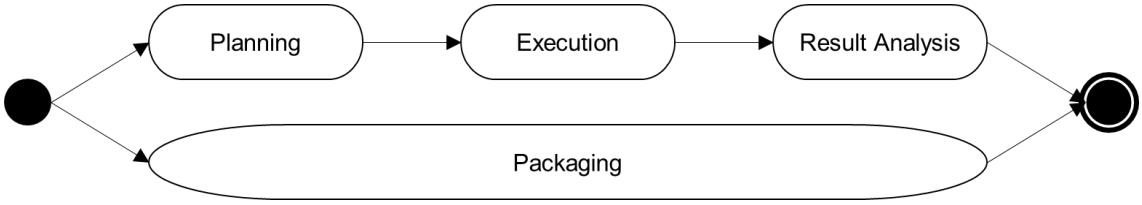

Figure 4.2 – SLR process (adapted from Biolchini et al. (2007))

- **Planning:** the first phase of the SLR process aims to confirm the need for conducting a systematic review. The central actions from this phase are the definition of the research question(s) and the definition of a research protocol, to set the procedures that will be adopted throughout the review. This protocol is then evaluated by an independent researcher, to ensure its quality;
- **Execution:** this phase involves the proper review, including the iterative evolution of the search string, to improve its precision and increase its coverage. Before the review execution, the researchers must define the

search strategies, including preliminary searches and evaluation of the results;

- **Result analysis:** after the paper selection and exclusion of those not attending the inclusion criteria predetermined, the selected papers are entirely read by the researchers, and a quality assessment is conducted, to evaluate how well each paper answers the research questions formulated for the SLR. The researchers then extract the data from the papers, the results are synthesized, and the research questions are answered;
- **Packaging:** throughout the SLR process, the packaging intends to document every step of the review, including decisions and information collected by the researchers.

The following subsections will discuss the phases mentioned above.

4.4 Review plan

4.4.1 Global and specific objectives

This study's objectives were stated according to the goal-question-metric (GQM) paradigm (VAN SOLINGEN et al., 2002), proposed by Basili and Rombach, and discussed with details in section 3.3.1.

The global objective of this study is to identify, in the technical literature, the technologies created to support the TDM. By "technology" we are considering any method, technique, tool, procedure or paradigm used in software development or maintenance, following the definition of Pfleeger (1999).

The specific objectives are presented below:

- **Analyze** technologies supporting TDM;
- **With the purpose of** characterizing;
- **With respect to** its applicability of use (context information as well as potential advantages and limitations);
- **From the point of view of** software engineering researchers;
- **In the context of** the software development process.

4.4.2 Research questions

To achieve the global and the specific objectives of this study, we formulated the central question and the following sub-questions:

- **RQ1:** Which software technologies have been proposed in the technical debt literature to support the technical debt management in the context of software projects?
    - **RQ1.1:** Which TDM activities are covered by those technologies?
    - **RQ1.2:** Which TD types are supported by those technologies?
    - **RQ1.3:** What is the software context in which those technologies should be applied?
    - **RQ1.4:** Which other technologies, from the Software Engineering context or other domains, are necessary to apply those technologies?
    - **RQ1.5:** Is there evidence on the use or applicability of those technologies?

4.4.3 Search string construction

Before the search string construction, a control reference list was compiled using some related secondary studies on TDM. These secondary studies were obtained in the initial *ad hoc* review of this work. The list of secondary studies is presented from now on:

- Ampatzoglou, Areti, et al. "The financial aspect of managing technical debt: A systematic literature review." Information and Software Technology 64 (2015): 52-73.
- Li, Zengyang, Paris Avgeriou, and Peng Liang. "A systematic mapping study on technical debt and its management." Journal of Systems and Software 101 (2015): 193-220.
- Alves, Nicolli SR, et al. "Identification and management of technical debt: A systematic mapping study." Information and Software Technology 70 (2016): 100-121.
- Fernández-Sánchez, Carlos, et al. "Identification and analysis of the elements required to manage technical debt using a systematic mapping study." Journal of Systems and Software 124 (2017): 22-38.
- Besker, Terese, Antonio Martini, and Jan Bosch. "A systematic literature review and a unified model of ATD." Software Engineering and Advanced Applications (SEAA), 2016 42nd Euromicro Conference on. IEEE, 2016.

- Ribeiro, Leilane Ferreira, et al. "Decision Criteria for the Payment of Technical Debt in Software Projects: A Systematic Mapping Study." ICEIS 2016 (2016): 572.

The 184 primary studies referenced by the secondary studies were submitted to the inclusion/exclusion criteria (explained later in this section). 70 studies came out of this selection, being adopted as a control list, providing some reference during the search string construction.

Initially, the first trial search string was created using the **PICO** strategy (PAI et al., 2004). The acronym stands for the patient (or population), intervention, comparison and outcome, each word representing one part of the search string. However, being this literature review with the purpose of characterizing technologies supporting TDM, it is not possible to obtain the "comparison" part of the PICO strategy. Thus, our study characterizes as a *quasi*-systematic literature review (*q*SLR) (TRAVASSOS et al., 2008). Therefore, the first trial of the search string was structured considering the following terms:

- **Population:** technical debt
- **Intervention:** software projects and technical debt management activities
- **Outcome:** software technologies

The first search string was executed by November 18th, 2017 and returned 67 papers. The search string passed through a series of trials; each trial is compared to the reference list primary studies from the secondary studies previously discussed. The goal was to adjust the search string to achieve the high level of coverage compared to the reference list of primary studies. It was observed that removing the "outcome" from the search string resulted in a higher coverage level compared to the control list. The last trial was executed by December 13th, 2017 and returned 356 papers.

Table 4.10 shows the search strings for the first and last trial. The use of the expression "TITLE-ABS-KEY" refers to the search only on the title, abstract and keywords. The use of an asterisk with some words (e.g., "repay*") is accepted by the *Scopus* search engine and is associated with every reference of that part of the word. For instance, both "repayment" and "repaying" are acceptable words in the search engine for the expression "repay*." Both the first and last trials are indicated in Table 4.10.

Table 4.10 – Search strings for the first and last trial

First trial *November 18th, 2017* 67 papers returned Control papers included in this trial: 32	TITLE-ABS-KEY (("technical debt" OR "design debt") AND ("software project*" OR "project of software" OR "software engineering" OR "software development" OR "development of software") AND ("identification" OR "measurement" OR "prioritization" OR "monitoring" OR "representation" OR "documentation" OR "prevent*" OR "repay*") AND ("practice*" OR "technique*" OR "method*" OR "process" OR "strateg*" OR "approach*" OR "model" OR "framework" OR "echnolog*"))
Last trial *December 13th, 2017* 356 papers returned Control papers included in this trial: 62	TITLE-ABS-KEY(("technical debt" OR "design debt") AND ("manag*" OR "identif*" OR "measur*" OR "quantif*" OR "assess*" OR "evaluat*" OR "estimat*" OR "calculat*" OR "prioritiz*" OR "monitor*" OR "represent*" OR "document*" OR "record" OR "register" OR "prevent*" OR "predict*" OR "repay*" OR "pay*" OR "eliminat*" OR "minimiz*" OR "reduce" OR "reduction" OR "optimiz*"))

After the last trial, eight papers from the control list did not return from the search. Those papers are not included in the *Scopus* database or, if they are indeed included, they do not have an abstract registered. Thus, those eight papers were included manually in our literature review, to be analyzed in the following phases.

4.4.4 Source and studies selection

The criteria to select the primary studies involves selecting the search engine and defining a set of inclusion and exclusion criteria. The *Scopus* database was selected, as it covers most of the software engineering publications.

For each of the 356 results on the last search trial, the researchers conducted the reading of title and abstract, proceeding to the selection of papers for a full reading. The set of inclusion and exclusion criteria is presented below:

- **Inclusion criteria:** these inclusion criteria should be considered assessed as follows: (IC1) and (IC2) and (IC3) and (IC4) and (IC5) and (IC6).
    - (IC1) It is related to software engineering AND
    - (IC2) It is explicitly about technical debt AND
    - (IC3) It proposes and provides information about a software technology that supports technical debt management AND
    - (IC4) Only English papers will be considered AND
    - (IC5) Gray literature (white papers) AND
    - (IC6) Available books chapter.
- **Exclusion criteria:** These exclusion criteria should be considered assessed as follows: (EC1) or (EC2) or (EC3) or (EC4) or (EC5) or (EC6) or (EC7) or (EC8) or (EC9) or (EC10) or (EC11).
    - (EC1) It is not about technical debt OR

- (EC2) It is conference proceedings, tutorials or extended abstract OR
- (EC3) It is a dissertation, thesis or technical report OR
- (EC4) It is a duplicated paper OR
- (EC5) It is a full book OR
- (EC6) It is a paper which has an evolution in another paper, in this case only the most completed will be considered OR
- (EC7) It does not propose or provide a software technology that supports technical debt management OR
- (EC8) It is not written in English OR
- (EC9) It is not related to software engineering OR
- (EC10) It is not available, even after being requested by the authors OR
- (EC11) It is a secondary or tertiary study.

The selection of studies followed the steps presented from now on:

- In the first evaluation, both researchers involved in this study read the title, abstract and keywords of all papers found in the search engine;
- Each study was evaluated by each researcher based on the inclusion and exclusion criteria, and classified into:
  - (I) included when the researcher considers this study should be selected for full reading;
  - (E) excluded if the researcher considers the paper fitting into one or more of the exclusion criteria. In that case, the researcher should indicate the criterion chosen for the exclusion;
  - (D) doubt, if he considers that the information on the title, abstract and keywords was not enough to assess the inclusion or exclusion;
- The individual evaluations were then compared, to conclude on the inclusion or exclusion of each paper. For that decision, the following considerations were taken:
  - If both researchers include or exclude the paper, then there was no further discussion. Included papers were selected for full reading;
  - If one or both the researchers had doubts on the paper, or if there was no immediate consensus on the inclusion or exclusion of the paper, then that specific paper was discussed among the researchers, and a final decision was made;

    - After the full reading, if the researchers found that any paper should be excluded, the exclusion would then be registered, along with the rationale;
- For the included papers on the previous step, the snowballing process (WOHLIN, 2014) was conducted, both forward and backward. In the backward process, one researcher was responsible for checking the references presented in the selected paper and selecting for extraction those in which title seems appropriate for the goal of this study. The forward process was conducted using the *Scopus* and *Google Academic* database, in which one researcher checked if another cited each selected paper. The title of each paper that cites one of the selected studies was assessed according to the goal of this study, being selected if appropriate;
- Finally, the selected studies were submitted for data extraction. This step was conducted during the full reading.

4.4.5 Information extraction strategy

The extraction phase aids the capture of information that can support the answer to the research questions. The form presented in Table 4.11 was created to support this phase. Both researchers conducted the full reading of the papers. Each researcher extracted the data of half the included papers, then reviewing the extraction from the other researcher for the other half of the included papers.

Table 4.11 – Data extraction form

Extraction field	Description
Reference information	Title, authors, year, source and abstract
Proposed technology for TDM	Name and description of the primary goal of the proposed technology
Context	Project type, development methodology, software domain
Other used technologies	Necessary technologies to apply the technology reported in the study. Technologies from the Software Engineering or other domain, e.g., financial domain
TD types	TD types that are supported by this technology, according to the types described by Alves et al. (2016)
TDM activities	TDM activities covered by this technology, according to the activities described by Li et al. (2015)
Evaluation type	Procedure to support the evaluation of the proposed technology
Other observations	Remarks regarding the benefits and limitations of the proposed technology. Other information the researcher seems fit to register.

4.4.6 Quality assessment criteria

The papers selected for full reading and data extraction were also submitted to a quality assessment, to evaluate the relevance of the studies and their adequacy to the goals of this research. Table 4.12 presents the questions of this quality assessment.

Table 4.12 – Quality assessment criteria

Category	Question	Score
Purpose	Q1 - Is the technology use purpose mentioned?	No - 0 pt It is possible to infer this information - 0,5 pt Yes - 1,0 pt
TD and TDM classification	Q2 - Is it possible to identify which proposed technology covers TD type?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
	Q3 - Is it possible to identify which TDM activity is covered by the proposed technology?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
Context	Q4 - Is it possible to identify whether the technology requires the use of other specific technologies?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
	Q5 - Is it possible to identify which project context (i.e., development process) is necessary to apply the proposed technology?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
Advantages	Q6 - Are the advantages of using this technology mentioned?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
Limitations	Q7 - Are the technology limitations discussed (threats to validity, application restrictions, etc.)?	No 0 pt It is possible to infer this information 0,5 pt Yes 1,0 pt
Evidence	Q8 - What kind of evidence supports the results of the studies that use this technology?	No evidence 0,0 pt Evidence obtained using a “toy” example 0,2 pts Evidence obtained through specialist opinions 0,4 pts Evidence obtained through academic studies (e.g., lab or classroom experiments) 0,6 pts Evidence obtained through industry studies (e.g., case studies) 0,8 pts Evidence obtained through consolidated industry practices 1,0 pt
	Final Score	

4.5 Execution and results

As previously mentioned, the last trial was executed by December 13th, 2017 and returned 356 papers. Figure 4.3 describes the selection process, from the last search trial on the *Scopus* database to the final study selection. In total, 77 studies were selected in this literature review, and their data were extracted using the form described in Table 4.11.

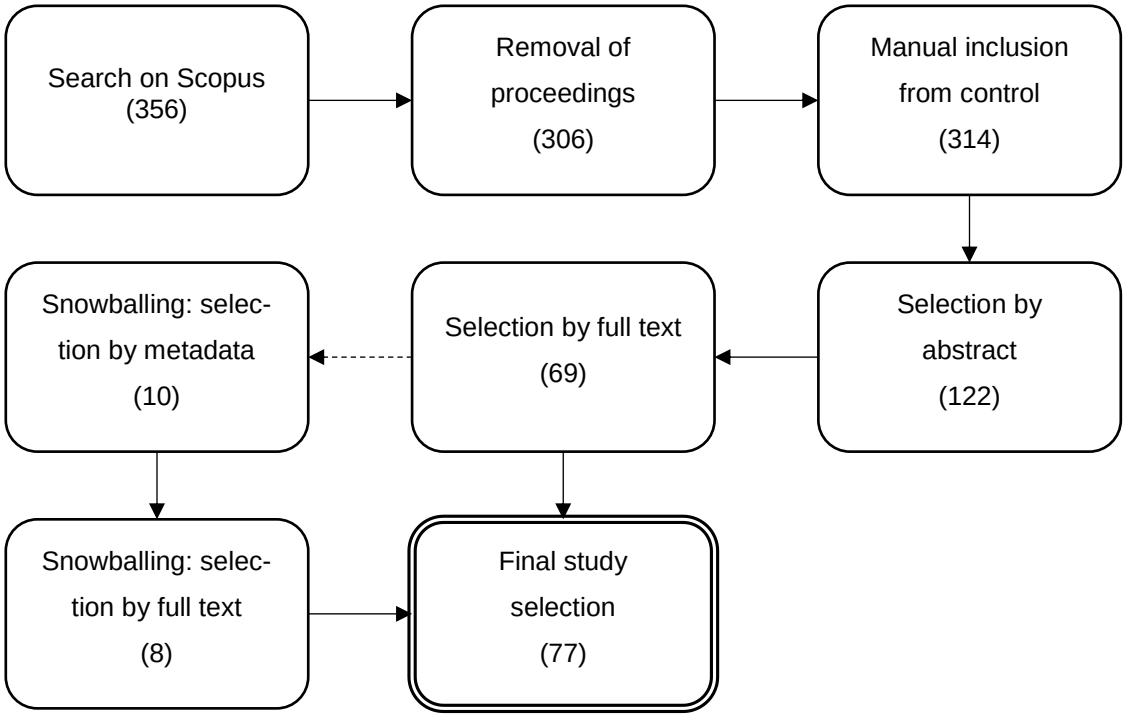

Figure 4.3 – Results of the search process

The following subsections present the results obtained from the studies. Additional information regarding demographics, sources, quality assessment and the list of studies for each category that was analyzed is presented in Appendix C of this dissertation.

4.5.1 Demographic results

Classification by publication type and source

The studies' sources were classified as either conference, workshop, journal, book chapter or symposium. Two studies could not be included in any of the categories above and are indicated with the tag "Other." The distribution of the studies on each type of source is indicated in Figure 4.4.

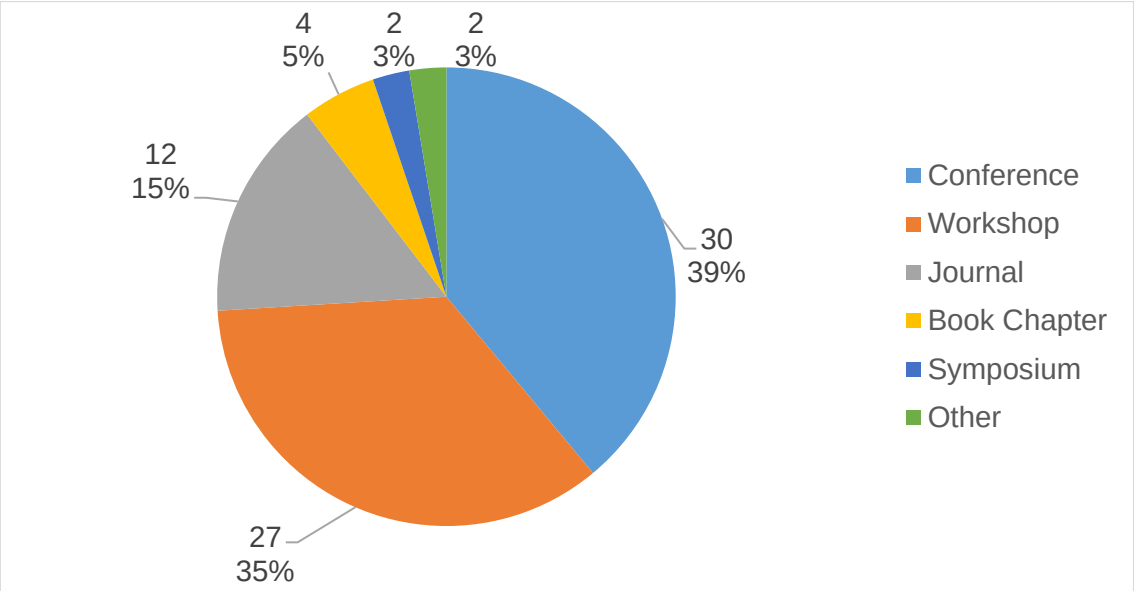

Figure 4.4 – Distribution of studies by source

The studies are distributed over 47 different publication sources. The most common source is conference (39%), followed by workshops (35%), journals (15%), book chapters (5%), symposiums (3%) and others (3%). Appendix C includes the detailing of the sources.

Classification by publication year

The studies obtained from the literature review were published between 2011 and 2018, being the year 2015 the one with most publications on the subject. Figure 4.5 presents the distribution of studies by the publication year and by the source type. It is possible to observe an increase in the studies from 2015 onwards. Comparing with the results from Li et al. (2015), there is a difference in the starting point of this increase, as the mapping study reported by the authors observed an increase of the selected studies starting in 2009 and 2010.

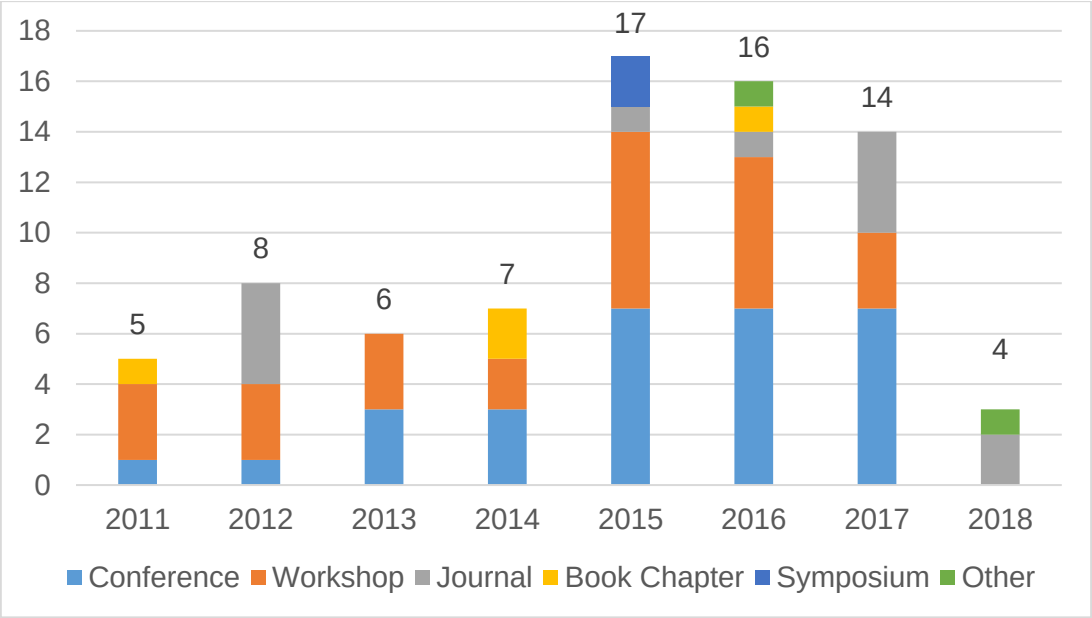

Figure 4.5 – Distribution of studies by year and type of source

4.5.2 Quality assessment

The quality assessment aims not to evaluate the technology *per se*, but to measure how much a study can answer the research questions raised in section 4.4.2, (KITCHENHAM, 2007). A total of eight questions were created to evaluate this, as described in Table 4.12. For each question, a score was assigned, between zero and one point. Therefore the total score in a study can receive is between zero and eight points. Despite also being used to weight the importance of individual studies during the data analysis (KITCHENHAM, 2007), this evaluation is not being performed in this study.

Like the data extraction, the studies were divided among both researchers for the quality assessment, being revised by the other researcher. The details on the quality assessment for each study is presented in Appendix C. The mean score is 5.15 out of a maximum of 8.00. It indicates that the average quality of the studies is not ideal, as it indicates that an appropriate amount of the information intended to be extracted from the studies are not available (as discussed with more details on some of the following sections). Regardless, the available information on the studies can still bring light to some of the topics that are part of our study's objectives.

Figure 4.6 presents the distribution of the studies over the scores. 45% (35) of the studies were evaluated in the range (5.0, 6.0], while 13 studies (17%) were in the highest range (6.0, 7.0]. 19 studies (25%) were in the range (4.0, 5.0], while eight studies were in the range (3.0, 4.0] and two studies were in the lowest range (2.0, 3.0]. The highest score assessed was 7.0 for study S14. The lowest score assessed was 3.0 for studies S52 and S63.

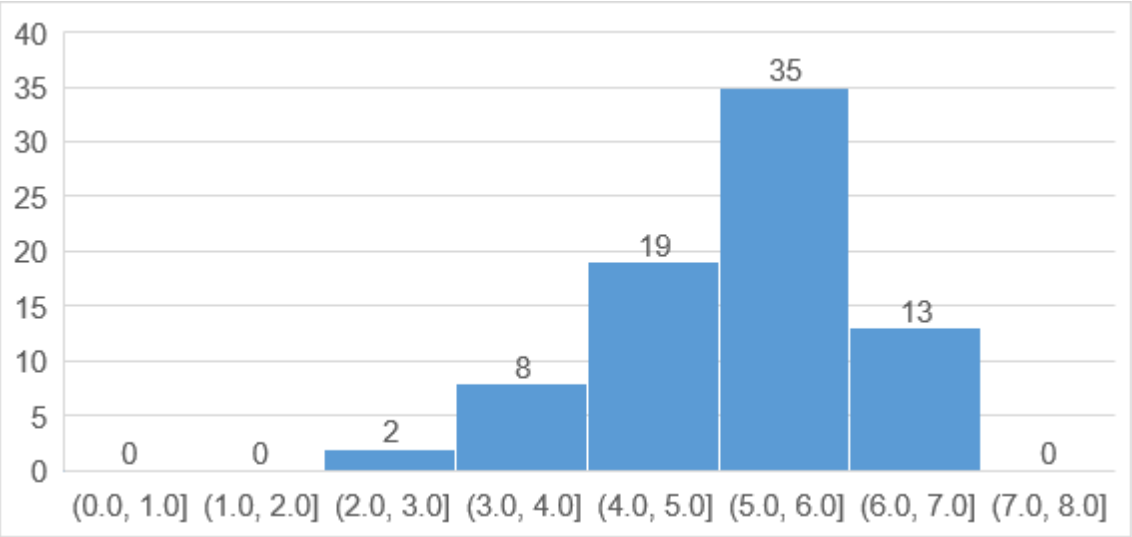

Figure 4.6 – Distribution of the selected studies over the quality assessment scores

4.5.3 TDM technologies

This section answers RQ1, “Which software technologies have been proposed in the technical debt literature to support the technical debt management in the context of software projects?”. A total of 77 technologies were identified, with different backgrounds and to apply on different artifacts. Appendix C presents a brief description of each technology.

The artifacts used by the technologies were registered and are presented in Figure 4.7. The source code was the most common artifact, present in 47 (61%) of the technologies. 13 technologies (17%) can be applied to any artifact in the software project. Six other artifact categories were found in 16 technologies (21%). One study did not have enough information to extract this data.

4.5.4 TDM activities

This section answers RQ1.1, “Which TDM activities are covered by those technologies?”. Figure 4.8 presents the number of studies for each of the eight TDM activities: identification, documentation/representation, measurement, prioritization, communication, monitoring, prevention, and repayment. 38 technologies (49%) are not specific to only one activity, being associated with two or more. The most represented activity is TD measurement, with 43 technologies (56%), followed closely by TD identification, with 42 technologies (55%). TD communication is the least frequent activity identified in the selected studies, showing up in 6 technologies (8%).

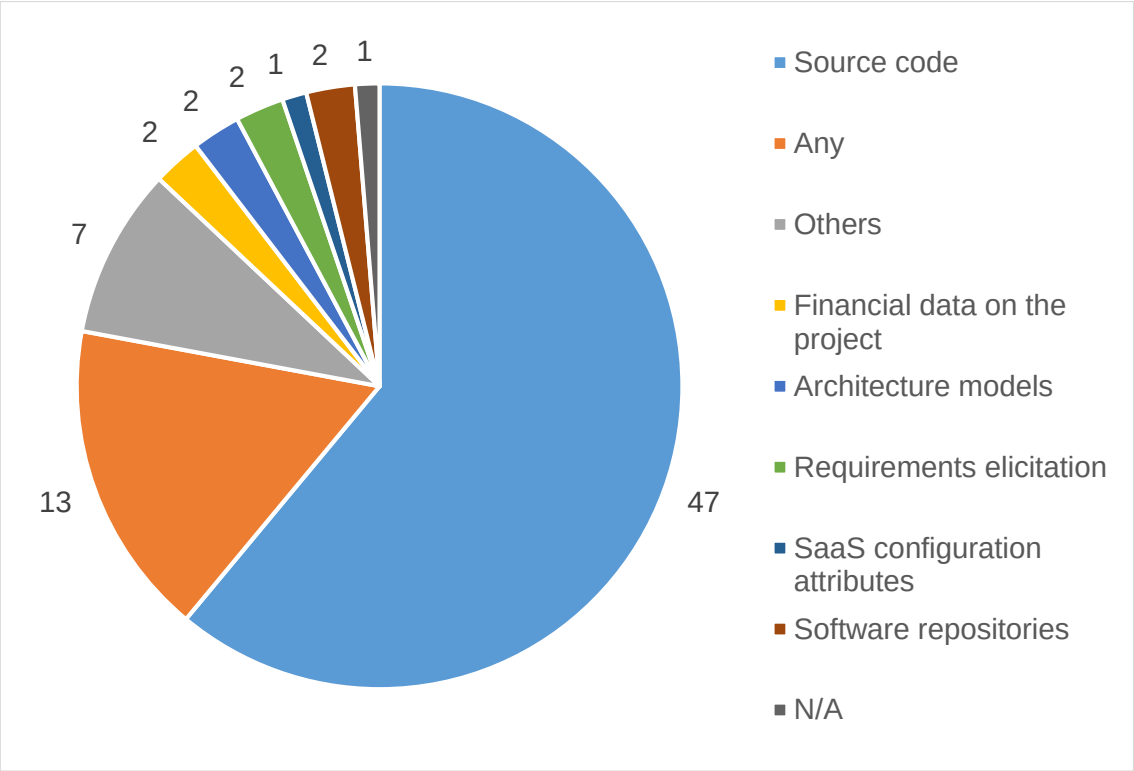

Figure 4.7 – Distribution of central artifacts applications for the selected studies

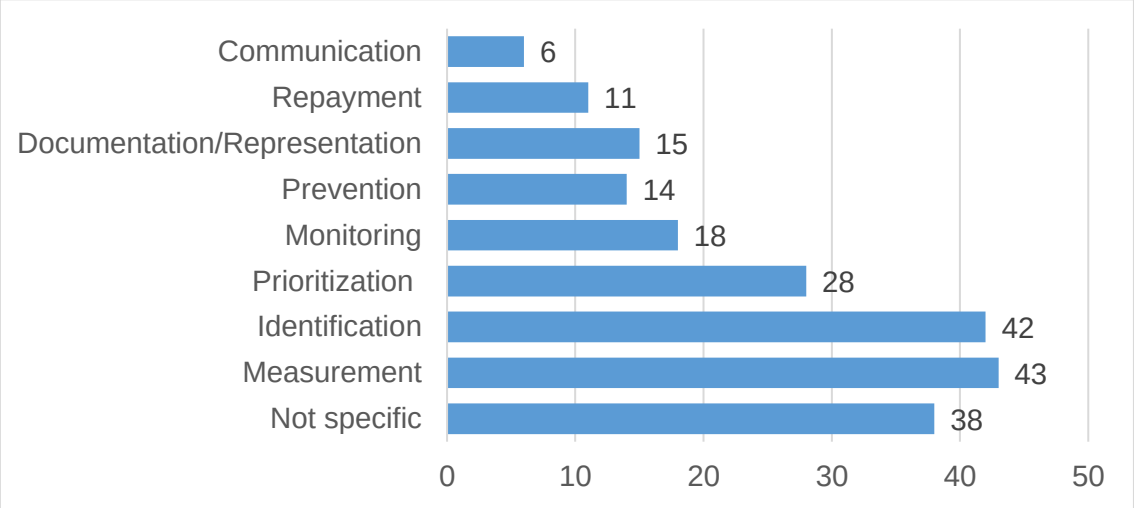

Figure 4.8 – Amount of studies related to each TDM activity

Figure 4.9 presents the same distribution in Figure 4.8 but separated by the technology backgrounds. It is possible to observe that most of the software metrics-related technologies are associated with TD identification and TD measurement. Some backgrounds, such as text mining and search-based techniques are only found on technologies associated with TD identification and TD measurement. Others, like software development practices and software quality assessment, are found on technologies associated with all or almost all TDM activities.

4.5.5 TD types

Answering RQ1.2, “Which TD types are supported by those technologies?”, the technologies were divided into TD types according to two classifications: intentional/unintentional debt (per McConnell (2007)) and by the TD taxonomy elaborated by Alves et al. (2016).

Considering the TD intentionality (Figure 4.10), 34 technologies (44%) are not specific to any type, 29 (38%) should be applied on unintentional debt, and 14 (18%) should be applied to the intentional debt.

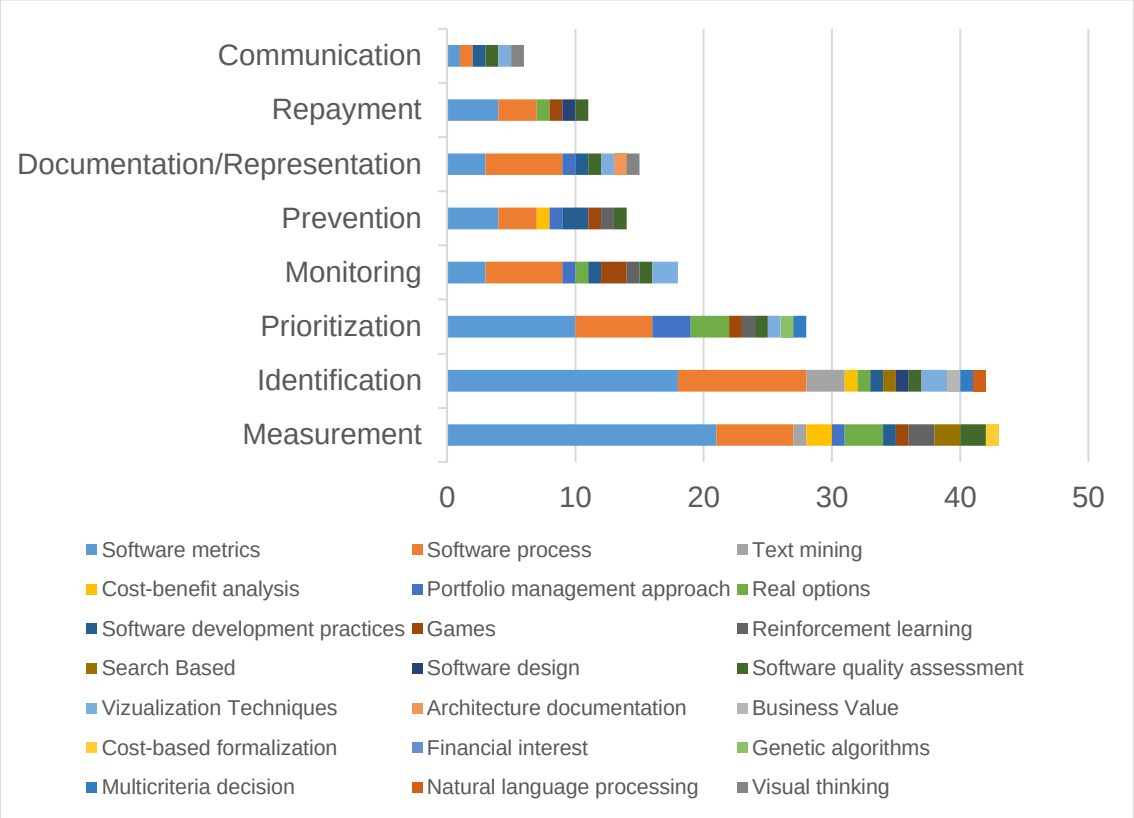

Figure 4.9 – Distribution of technology backgrounds over the TDM activities

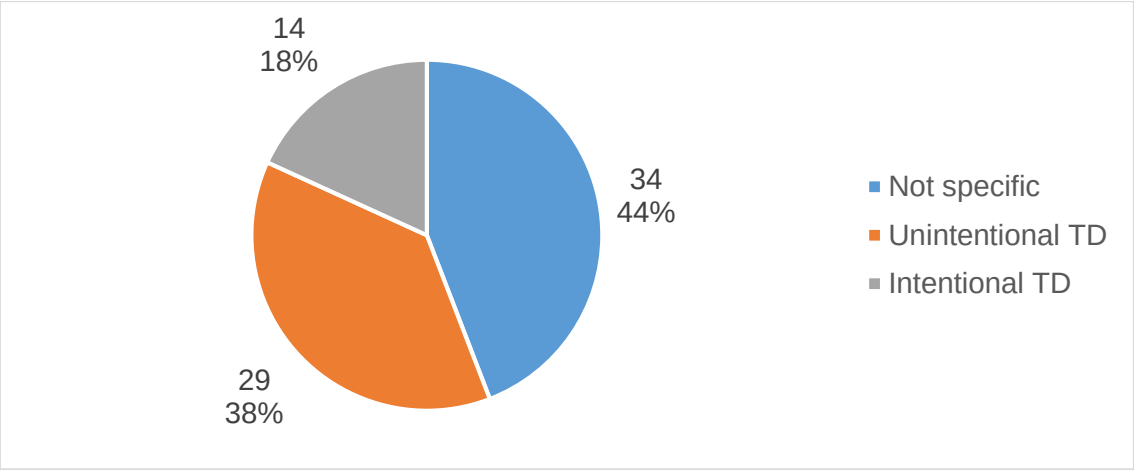

Figure 4.10 – TD types for each study, divided by intentionality (as per McConnell (2007))

44 technologies (57%) are no specific to a single type of TD, according to Alves et al.'s (2016) taxonomy, i.e., they are associated with at least two different TD types (Figure 4.11). It frequently occurs in code-related technologies, as often they deal with code, design and (sometimes) architecture debt. Those three TD types are also the most recurrent TD types, found in 50 (65%), 48 (62%) and 44 (57%) of the technologies, respectively. The least frequent TD types are build debt and versioning debt, both appearing in 10 (13%) technologies.

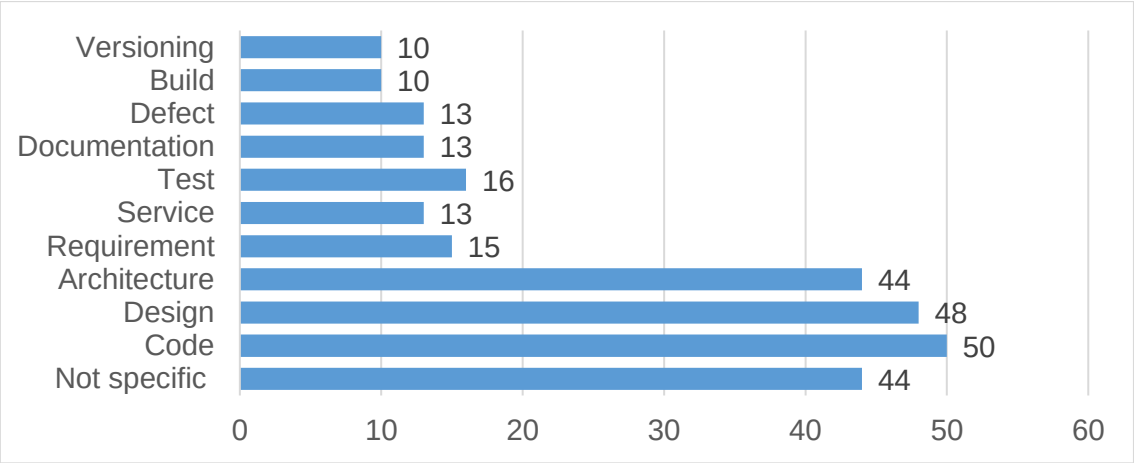

Figure 4.11 – TD types for each study, divided according to Ales et al.'s (2016) taxonomy

Finally, it is possible to analyze the relationship between the TD types and the TDM activities, as presented in Figure 4.12. The number into the bubbles represent the number of technologies that are associated with both the TD type and the TDM activity. For instance, 28 studies are related to both TD identification and code debt. For TD repayment, TD prevention, and TD communication, there is little association with types besides code, architecture, and design.

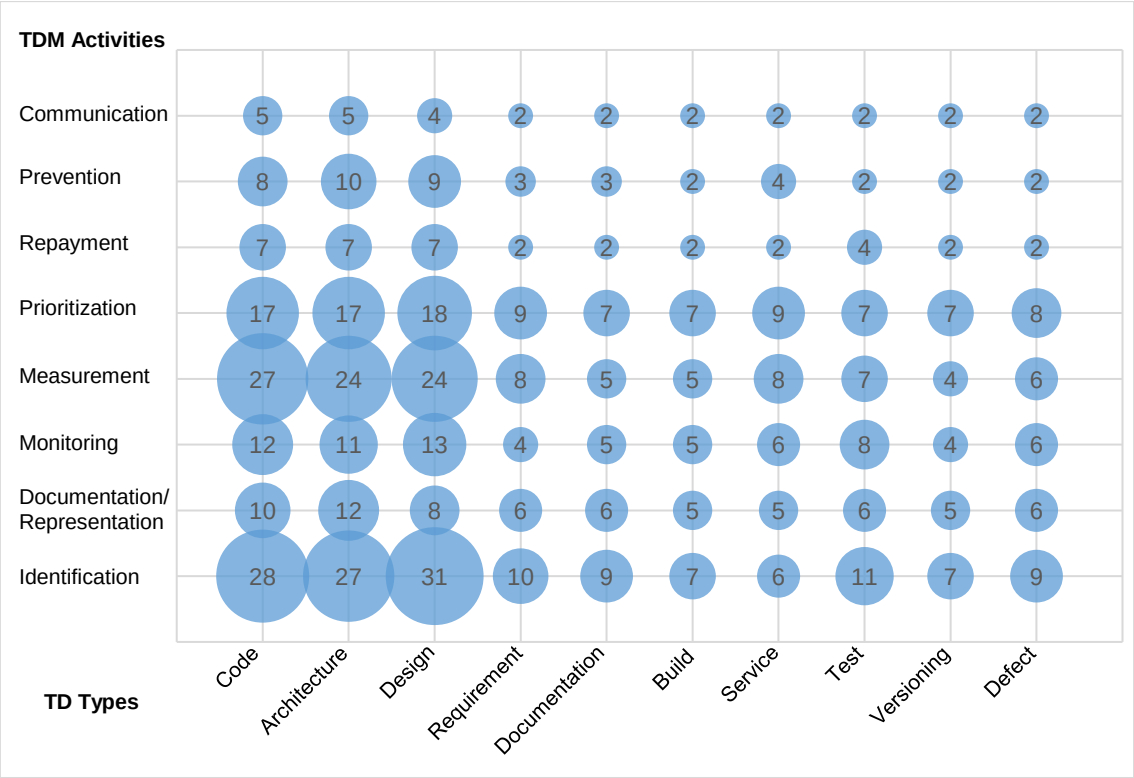

Figure 4.12 – Relationship between TD types and TDM activities

4.5.6 Software/Project context

To answer RQ1.3, “What is the software context in which those technologies should be applied?”, we considered the development model and the system or project domain registered in the evidence provided by the authors of the studies (Figure 4.13). We also registered the programming languages applicable to each selected technology.

63 studies (82%) were classified as independent of the development model, while 13 studies (17%) were associated with software developed using iterative or agile practices. Only one study was associated with the waterfall model.

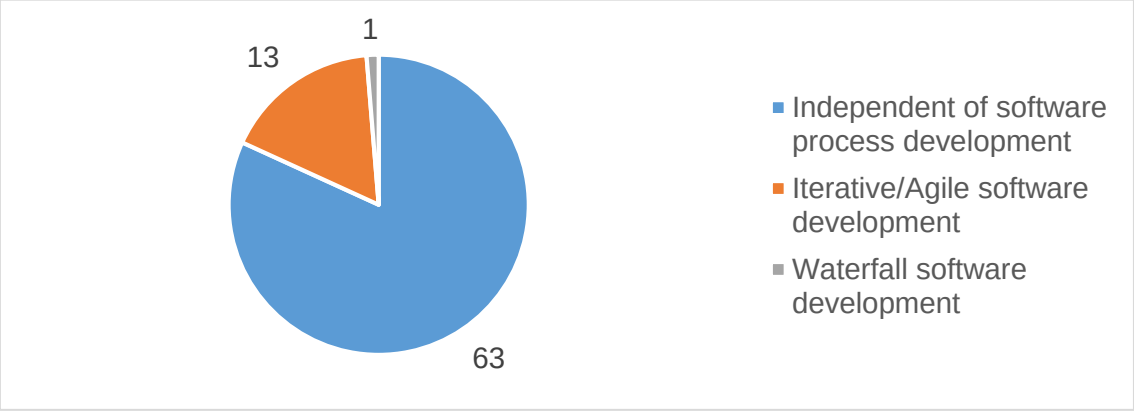

Figure 4.13 – Classification of studies according to the development model

As for the classification according to the system or project domain (Figure 4.14), 47 studies (61%) did not have enough information on the domain or did not present evidence to support their respective technologies. The remaining studies were distributed among 12 categories, being the most common the software development domain (9 technologies or 12%) and the least frequent the automotive, consulting, financial, government, health, oil & gas and social networks domains, each being represented in one study.

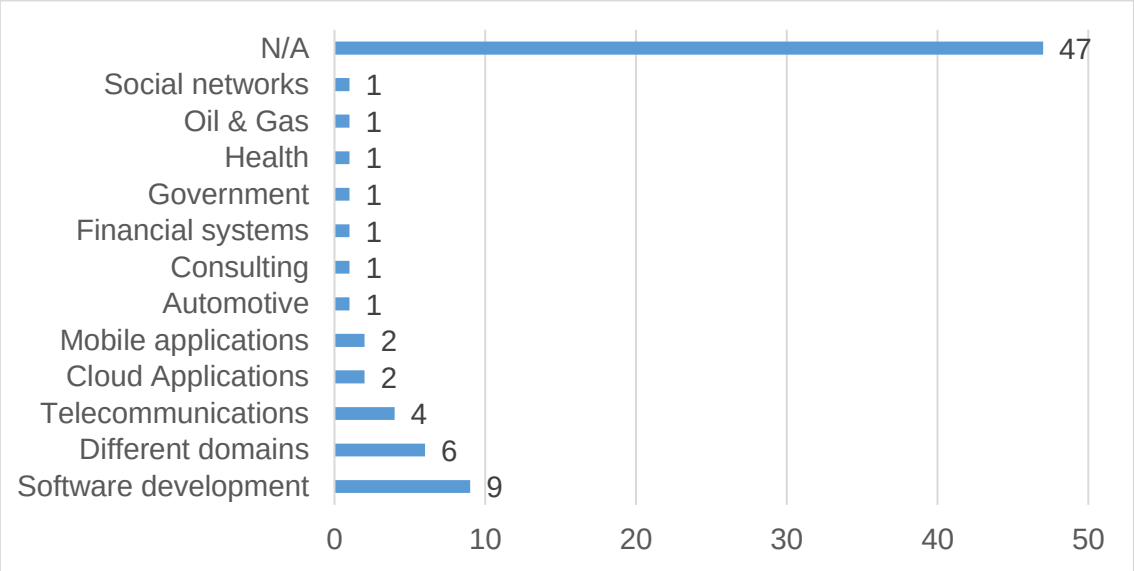

Figure 4.14 – Classification of studies according to the system or project domain

Finally, we investigated each programming language that could be used alongside each selected technology (Figure 4.15). 21 studies (27%) reported not being related to any specific programming language, while 20 studies (26%) reported technologies applicable only to Java. Three studies (4%) reported technologies that could be applied to multiple programming languages, and 28 (36%) studies did not present any information on whether the technology depends on any specific programming language.

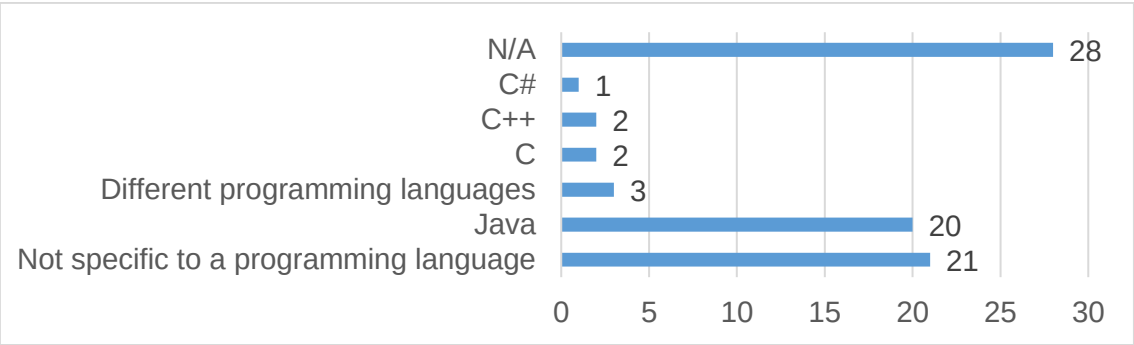

Figure 4.15 – Distribution of the studies according to the programming language applicable to each selected technology

4.5.7 Dependence on other technologies

This subsection answers RQ1.4, “Which other technologies, from the Software Engineering context or other domains, are necessary to apply those technologies?”. As presented in Figure 4.16, most of the technologies (32 or 42%) depend on third-party tools, those being, for example, Eclipse (5 technologies or 6%). 27 studies (35%) report the use of proprietary tools. 15 studies (19%) did not report any dependence or did not provide enough information on the technology to enable this particular information extraction.

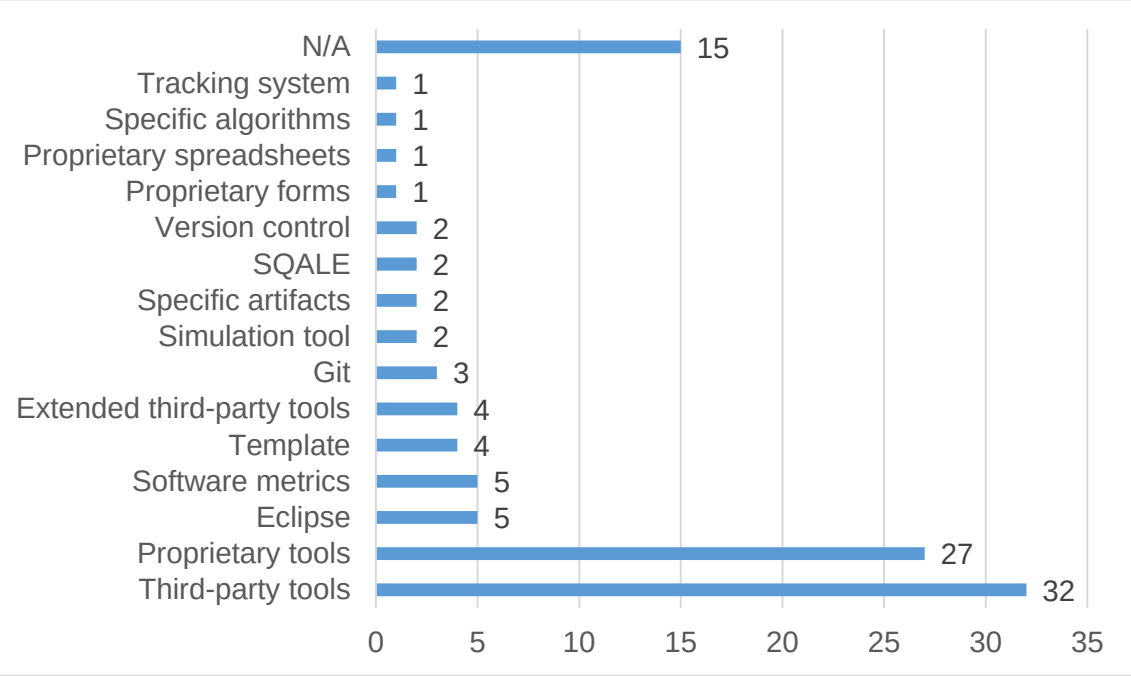

Figure 4.16 – Dependence from other technologies

4.5.8 Evidence from the studies

Finally, this subsection answers RQ1.5, “Are there evidence on the use or applicability of those technologies?”. Figure 4.17 presents the data extracted. As previously discussed on section 4.4.5, the evidence was divided into six types, from weaker to stronger evidence: no evidence, toy example (or proof-of-concept), specialist opinions, lab or classroom experiments, case studies and industry practices.

11 studies (14%) did not report any evidence on their technologies. The most frequent type of evidence was toy examples, present in 22 (29%) studies. Specialist opinions were the least common type of evidence, present in only 2 (3%) studies.

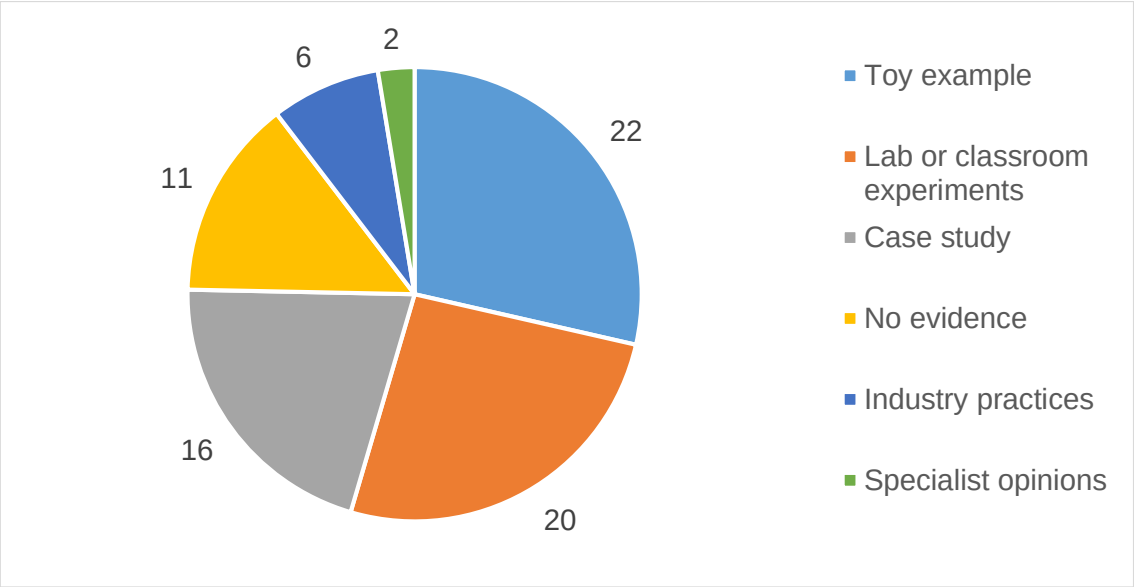

Figure 4.17 – Distribution of selected studies over the type of evidence

For this research question, in particular, it is important to emphasize that the literature review presented conditions to assess only if the technology has evidence in its original study. There are cases, not mapped in this literature review that the technology is further discussed in later publications, presenting stronger evidence on the applicability of the said technology. It is the case, for example, of the *DebtFlag* tool, in which is presented in (HOLVITIE; LEPPANEN, 2013) with no evidence and is appropriately evaluated in (HOLVITIE, 2014).

4.6 Threats to validity

The results of this literature review may be affected by some threats of validity. Those were discussed among the researchers, and the actions were taken to mitigate the possible consequences of the threats. Among of those threats, it is possible to include the limitation on the identification of primary studies, the researchers' bias on the study selection, the quality assessment, and the data synthesis, and the inaccuracy of the data extracted from the studies. Those topics are discussed in the following subsections.

4.6.1 Identification of primary studies

The main threat identified on this topic is related to the decision to choose *Scopus* as the only search engine. It could limit the search results of the literature review. Two actions were taken to mitigate this threat:

- A control list was created, using as reference a list containing the results from previous secondary studies related to TDM. The results of this list

that were not found on the *Scopus* database, or were in the search engine but did not return using the final search string, were manually added to the selection list;

- The forward and backward snowballing technique was adopted, to cover studies mentioned by the selected list and those that cite any study present on the selected list. It was especially relevant to include the papers published after the search string was executed.

4.6.2 Researchers' bias

Several steps of the literature review include the critical analysis of the researchers, either to select the studies by their abstract, to assess their quality or to synthesize their data after extraction. This bias can result in misleading results, absent studies on the final list, and misclassification of the final data. Some actions were taken to reduce this threat:

- All tasks were divided between two researchers, and the information one researcher produced was revised by the other. It occurred throughout all the review process;
- All doubts raised during the study selection and data synthesis were explicitly registered and discussed among the researchers. During the study selection phase, whenever a doubt raised, the study was included for further analysis in the next step, until the doubts were eliminated after a consensus among the researchers;
- The data synthesis passed through some cycles of text revision by the researchers while the results were being analyzed until both researchers came to a consensus on the information synthesized.

4.6.3 The inaccuracy of data extraction

Finally, the data extracted from the selected studies could have been misinterpreted or misclassified during the data extraction step. The number of selected papers was considerable, and some errors could have happened. The data extraction forms filled by one researcher were revised by the other, to observe any missing information or incompatibility between the data extracted and the source material to mitigate this potential threat

4.7 Chapter considerations

Despite this literature review focusing on a different point-of-view than those of previous secondary studies on TD and TDM, some similarities could be used as a vali-

dation of our study and as an indication of the topic's evolution in the academic community.

First of all, it is essential to observe that the number of studies on TD is increasing in the past years. Tom et al. (2013) reported in 2013 a multivocal literature review with a broad scope, using simple terms like "technical debt" or "design debt," and returned only 35 results. Later, in 2015 Li et al. (2015) conducted a systematic mapping study with a series of search engines, using the term "technical debt" and looking for papers discussing TD types or new concepts, with or without TDM technologies, and returned 94 studies on their final selection. Our literature review, with a much more specific scope (only TDM technologies), returned 77 studies on the final selection. It indicates an increasing interest in the community about TD and how to manage it properly.

Furthermore, our study is coherent to the results found by Li et al. (2015) on the relation between TD types and TDM activities. Their comparison (reproduced in Figure 4.18) revealed some research opportunities that were at least addressed by the community in recent studies, according to our results. For example, study S28 addresses TD prevention and documentation debt, which was a field previously identified as a research gap by Li et al.

Our results also corroborate with other types of studies. Ernst et al. (2015), for instance, affirmed that most of the used tools to manage TD were issue trackers, which are based mostly on software metrics and related to the source code, revealing code, design and (occasionally) architecture debt. Our study indicated that those three types of debt are the most common on current TDM technologies, and the two most ordinary activities are TD measurement and TD identification, that frequently use issue trackers or similar tools to manage the TD.

Despite that, we found some incoherence with other studies. For example, Yli-Huumo et al. (2016) concluded in their industrial survey that TD measurement was rarely used. Despite that, it is the most frequent TDM activity associated with the technologies revealed in our study. It can indicate a mismatch between the academic understanding of the industrial needs and the real necessities of the software practitioners.

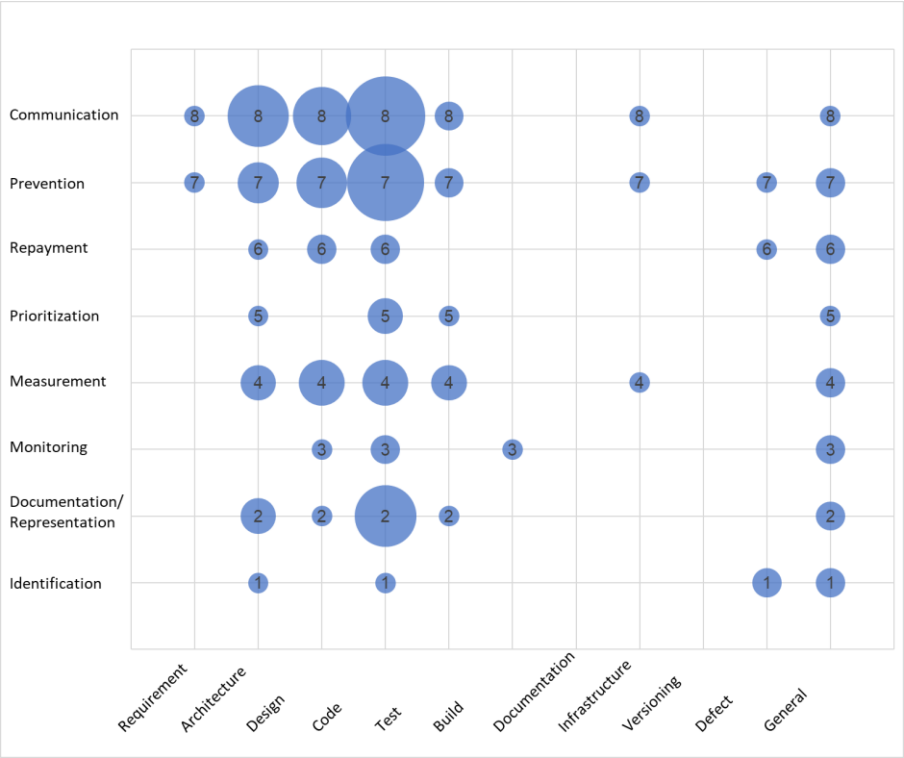

Figure 4.18 – Analysis between TD types and TDM activities by Li et al. (2015)

Regarding the dependence of other technologies, our study selection indicates that several TDM technologies depend on third-party tools (like Eclipse) or are an extension of those, such as plugins. It occurs in 32 of the 77 selected studies. Other 27 technologies depend on proprietary tools developed by the respective authors or teams.

As contributions of our research to the academic and industrial communities, besides de confirmation of some information reported by other authors, it is worth mention:

- An extensive list of technologies, ranging from templates to sophisticated tools, to manage different types of TD under several conditions;
- The revelation of, possibly, over-researched topics, such as technologies for identification of code debt. On the other hand, our study also revealed some still present research gaps, with only a small amount of technologies, like technologies to cover building or versioning debt, or some activities to manage documentation and requirement debt;
- The revelation of the necessity to further evaluate the current TDM technologies, as only 29% of the studies presented evaluations of the technologies directly in the industry, like case studies or actual industrial practices.

5 An evidence briefing on technical debt management

The findings of the quasi-Systematic Literature Review and the results from the Survey were applied into building an artifact aiming to aid practitioners from the software industry to better understand the concept of technical debt and to apply the technologies revealed in the qSLR to manage it.

5.1 Introduction

The results of the survey presented in chapter 3 indicated a possible lack of knowledge of the Brazilian software practitioners regarding TD. Very few professionals knew what TD is. Out of those who affirmed knowing the concept, we found some misconceptions about their TD definition, when comparing to the concept evolved in academia. Moreover, even fewer practitioners affirmed adopting some practices to manage the TD. The number of practitioners who use some technologies and formal processes is even lower.

The use of simple practices to manage the TD in the industry was also observed by other researchers (ERNST et al., 2015; YLI-HUUMO; MAGLYAS; SMOLANDER, 2016). One of the reasons for that, according to Ernst et al. (2015), is a lack of proper tools to manage TD. However, among the results of the *q*SLR discussed in chapter 4, we can observe a reasonable number of technologies for TDM, 27 of them based on proprietary tools developed by researchers or their teams.

It is important to emphasize, though, that those technologies do not necessarily have immediate applicability to the practitioners, as some of them are still prototypes and their applicability should be further investigated. Notwithstanding, it is possible to assemble a list of technologies that can be used by the software practitioners.

Therefore, this chapter aims to provide an instrument to aid software practitioners to gain knowledge and to assist in disseminating the existing TDM technologies to practitioners in the software industry. To achieve this objective, knowledge acquired in this dissertation is going to be aggregated using the concepts of evidence briefings (EBs), as discussed by Cartaxo et al. (2016).

This chapter is structured as follows: section 5.2 details the concepts behind the evidence briefing, section 5.3 presents the requirements for the instrument to be

developed, section 5.4 describes the activities conducted by the author to develop the instruments, and section 5.5 provides some general considerations on this chapter.

5.2 Evidence briefing

The need of EBs as a medium to transfer knowledge from researchers to the industry comes due to the reality that software practitioners tend to not use research papers as a source of new knowledge (CARTAXO et al., 2016). Thus, it is necessary to develop more concise instruments, which summarize the ideas and main findings of the paper to a broader audience.

An EB, as described by Cartaxo et al. (2016) is a one-piece document that reports the main findings of empirical research. It combines elements of Information Design and Gestalt Theory to enable documents more appealing to the final reader and easier to read. The leading design of the EBs was created based on several other types of mediums proposed in medical research.

As the authors concluded after their studies, the use of EBs has several advantages, such as it can be used of an executive summary, it increases the research visibility, it was considered a better way to share findings, and it stimulates reading the original research material. Among their findings after empirical studies with researchers and practitioners, they found that the respondents considered EBs reliable, easy to find information and “clear and understandable information” (CARTAXO et al., 2016).

The main template, available to use under an open source license (CC-BY) in the link http://cin.ufpe.br/eseg/briefings, is created with some characteristics, as described below and represented in Figure 5.1:

- The title of the briefing (1), sometimes simplifying the paper title to make the briefing more appealing to the practitioners;
- A summary (2) to present the primary objectives of the briefing;
- The main findings of the briefing (3), extracted from the original empirical study;
- Informative box (4), separated from the main text, to highlight the target audience and the purpose of the briefing, as well as the primary references;
- The references to the original empirical study (5);
- The logos and identification of the research group and the university (6).

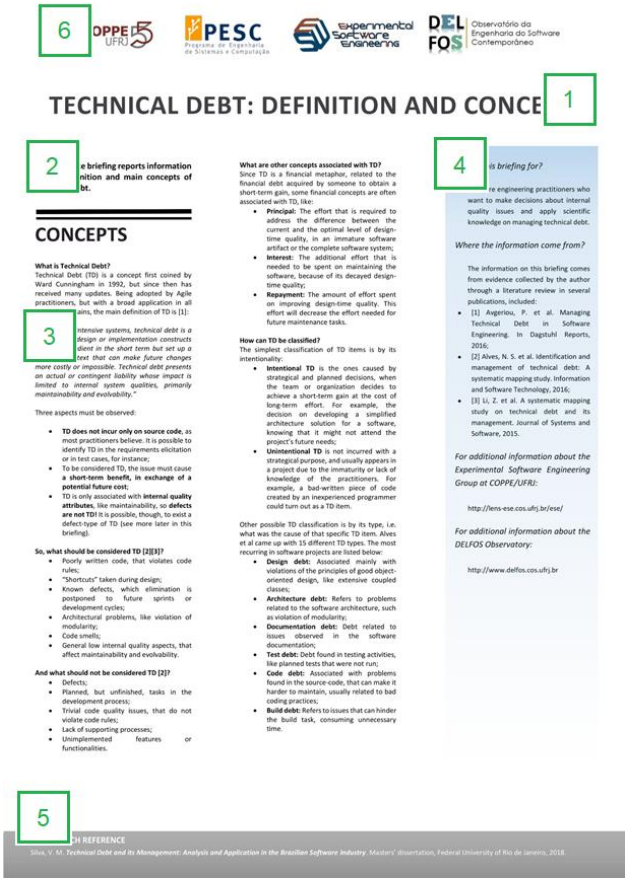

Figure 5.1 – Overview of the main elements of an EB

5.3 Requirements

This section aims to describe the requirements of the EBs that are going to be generated based on the findings of this dissertation. The primary requirements are related to the concept of TD, the TDM activities and the technologies to manage TD:

- **Req1:** The instrument must present the concept of TD, including some examples of what should be and should not be considered TD;
- **Req2:** The instrument must present the main types of TD, according to the taxonomy proposed by Alves et al. (2016). If necessary, the instrument can present only the most frequent TD types occurring in the technologies described in the EBs;
- **Req3:** The instrument must introduce the different TDM activities, alongside with their definitions and leading approaches to conduct each activity, as described by Li et al. (2015);
- **Req4:** The instrument must present the technologies to manage TD in each activity independently, indicating the following information (if available):

    - Name of the technology;
    - Brief description;
    - Source artifact;
    - TD types covered by the technology;
    - TDM activities supported by the technology;
    - Link to the technology, supplied by the authors of the studies.
- **Req5:** The technology must be available, i.e., all necessary instruments (tools, templated, processes, etc.) must be accessible by the time the EB is being created (June 2018);
- **Req6:** The technology must have been evaluated through experiments with the industry or a survey of expert practitioners, or presented in an experience report;
- **Req7:** The EBs must present the references to the studies that present the technologies, in case they are processes, methodologies or approaches in general;
- **Req8:** The instrument must have two versions, one in Portuguese and one in English.

5.4 Organization

5.4.1 Selection of the technologies

The initial selection of the technologies to be listed in the EBs was a compilation of the ones obtained through the *quasi*-systematic literature review (*q*SLR) co-conducted by the author of this dissertation (presented in chapter 4) and through the survey conducted with practitioners in the Brazilian software industry (presented in chapter 3). The literature review revealed 77 technologies, and the survey revealed seven additional tools and 15 additional practices, summing up 99 different technologies.

For the technologies revealed in the literature review, an analysis of the type of evidence and availability was conducted, containing the following activities:

- **Step 1:** All papers were wholly read one more time, searching for information on the current status of the technology and the necessity of specific tools to use the technologies;
- **Step 2:** If any proprietary tools were needed, additional data regarding possible download links or more information available online were extracted;

- **Step 3:** If no information of the kind was available, a search on the *Scopus* and *Google Academic* engines was conducted, looking for follow-up studies by the same authors regarding that technology;
- **Step 4:** If the original study presented a name for the technology, it was searched using the *Google* website to attempt obtaining institutional information, or a page of the technology;
- **Step 5:** For the studies that had for evaluation type in the *q*SLR "no evaluation," "toy example," "specialist opinions" or "lab or classroom experiments," the same search described in **step 3** was conducted, but looking for other types of evidence;
- **Step 6:** With all information collected, the technology was then classified according to the type of information extracted from the previous steps:
    - **Experimentally evaluated**, if the technology attends **req5** (described in section 5.3), i.e., if it has all the information necessary to be put in practice; and if it was evaluated through a case study or action research;
    - **Industry practice**, if the technology has all the information necessary to be put into practice and has an available supporting tool to be used, but was reported as an industry practice;
    - **Survey with practitioners**, for the technologies revealed in the survey with Brazilian software practitioners (see Table 3.6); and
    - **Excluded**, if the technology does fit into any of the previous classifications.

It is important to emphasize that, since the target audience of these EBs is the software practitioners, the usual step of contacting the authors to obtain information on the technologies is not being conducted, as this is not expected to be executed by professionals in the industry. Therefore, only technologies available for the final consumer should be considered.

Out of the 99 technologies, 54 were classified as "excluded," 22 were classified as "survey with practitioners," nine were classified as "industry practice," and 14 were classified as "experimentally evaluated." The 45 technologies classified as either "survey with practitioners," "experimentally evaluated" or "industry practice" was included in the EBs.

5.4.2 Design

One significant activity is the content organization in the EB. The information should be included in a way that is easy to read and understandable to practitioners

unfamiliar with the concept. Due to a vast amount of information, only one EB page was not sufficient. Therefore, we decided to divide the content into three EBs, discussed in the following subsections.

The content of each EB was elaborated in English; then the files were formatted to adjust the necessary information. Finally, the content was translated into Portuguese, to attend **req8**, as described in section 5.3. The three English-written pages are presented at the end of this chapter, while the Portuguese-written pages are presented in Appendix D of this dissertation.

5.4.2.1 "EB1 – Technical debt: Definitions and Concepts"

This EB (presented in Figure 5.2) includes the definition of TD, examples of TD and non-TD, financial concepts associated with the metaphor and most common TD types.

First, an introduction was made, discussing the creating of the TD definition and how it evolved into the current definition, proposed in the *Dagstuhl* seminar (AVGERIOU et al., 2016). Then some observations were made regarding TD, discussing some common misinterpretations or misuses of the concept:

- We discuss that TD does not incur only on source-code, being possible to identify it in other artifacts. This comment relates to the evidence revealed on chapter 4 and on other studies, that most technologies to manage the TD focus on code-related debt, and brings awareness to the presence of TD in other sources and other types of TD;
- We emphasize two of the main aspects of the TD definition: the issue that creates a TD item must create a short-term benefit, in exchange of a potential future cost, and the relation between TD and internal quality attributes. Those statements help to narrow the TD definition, excluding defects, which are related to external quality attributes, and other technical decisions that do not cause a short-term benefit to the project;
- For the Portuguese version of this first EB, we decided to include an observation to clarify the proper translation of "technical debt," being "dívida técnica" and not "débito técnico." The word "debt," translating to Portuguese, can mean either "dívida" or "débito," depending on the context, and having different meanings. Since the meaning of "débito técnico" differs from the idea of incurring in something the practitioner owns, we felt that this distinction was essential to be made in the EB.

After the introduction, two sections were created, presenting examples of what should and should not be considered TD. Those examples were adapted from Li et al.

(2015) and Alves et al. (2016). Following those sections, we included a paragraph with some common financial terms associated with the metaphor. Due to physical space reasons, we decided to include only “principal,” “interest” and “repayment” on the list, as they are essential to understanding most of the activities, technologies and other definitions presented on the EBs.

Finally, the classification of TD items is discussed, focusing on the intentionality, as per McConnell (2007), and on the taxonomy of TD types, as per Alves et al. (2016). Due to the lack of space, only design debt, architecture debt, documentation debt, test debt, code debt and build debt are described.

5.4.2.2 “EB2 – Technical debt management: Activities and Practices”

This EB (shown in Figure 5.3) includes the definition of the TDM activities described by Li et al. (2015) and presents the practices and guidelines to manage the TD. Those were gathered from the survey conducted with software practitioners in BSOs, presented in chapter 3, and from the qSLR conducted and presented in chapter 4 of this dissertation.

For common practices in the industry, such as backlogs or refactoring, only the name of the practice was reported, grouped in the TDM activities. For industry practices reported in the technical literature and obtained through the qSLR, a table was created, presenting a summary of the practice, the TDM activities covered by that practice, the TD types covered, the source artifact, the type of evidence included in the study and the paper reference.

5.4.2.3 “EB3 – Technical debt management: Tools and Strategies”

The final EB (presented in Figure 5.4) includes the tools, methodologies, strategies, approaches or frameworks to manage the different TDM activities. Similar to the previous one, tables are adopted to report the technologies.

A first table was created to report the tools. The table contains the name and a description of the tool, the link to download it, the TD types covered and the TDM activities that are supported by the tools. The source artifact is also presented, along with the type of evidence reported and the reference to the study associated with that tool.

The second table reports other technologies, such as methodologies or frameworks. This table has the same content of the previous one, except the “Link” column, which is not relevant, as the user of the EB must consult the reference paper to understand the technology better.

TECHNICAL DEBT: DEFINITION AND CONCEPTS

This evidence briefing reports information on the definition and main concepts of technical debt.

CONCEPTS

What is Technical Debt?
Technical Debt (TD) is a concept first coined by Ward Cunningham in 1992, but since then has received many updates. Being adopted by Agile practitioners, but with a broad application in all software domains, the main definition of TD is [1]:

"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability."

Three aspects must be observed:

- **TD does not incur only on source code**, as most practitioners believe. It is possible to identify TD in the requirements elicitation or in test cases, for instance;
- To be considered TD, the issue must cause **a short-term benefit, in exchange of a potential future cost**;
- TD is only associated with **internal quality attributes**, like maintainability, so **defects are not TD!** It is possible, though, to exist a defect-type of TD (see more later in this briefing).

So, what should be considered TD [2][3]?
- Poorly written code, that violates code rules;
- "Shortcuts" taken during design;
- Known defects, which elimination is postponed to future sprints or development cycles;
- Architectural problems, like violation of modularity;
- Code smells;
- General low internal quality aspects, that affect maintainability and evolvability.

And what should not be considered TD [3]?
- Defects;
- Trivial code quality issues, that do not violate code rules;
- Lack of supporting processes;
- Unimplemented features or functionalities.

What are other concepts associated with TD?
Since TD is a financial metaphor, related to the financial debt acquired by someone to obtain a short-term gain, some financial concepts are often associated with TD, like:
- **Principal:** The effort that is required to address the difference between the current and the optimal level of design-time quality, in an immature software artifact or the complete software system;
- **Interest:** The additional effort that is needed to be spent on maintaining the software, because of its decayed design-time quality;
- **Repayment:** The amount of effort spent on improving design-time quality. This effort will decrease the effort needed for future maintenance tasks.

How can TD be classified?
The simplest classification of TD items is by its intentionality:
- **Intentional TD** is the ones caused by strategical and planned decisions, when the team or organization decides to achieve a short-term gain at the cost of long-term effort. For example, the decision on developing a simplified architecture solution for a software, knowing that it might not attend the project's future needs;
- **Unintentional TD** is not incurred with a strategical purpose, and usually appears in a project due to the immaturity or lack of knowledge of the practitioners. For example, a bad-written piece of code created by an inexperienced programmer could turn out as a TD item.

Other possible TD classification is by its type, i.e. what was the cause of that specific TD item. Alves et al [2] came up with 15 different TD types. The most recurring in software projects are listed below:
- **Design debt:** Associated mainly with violations of the principles of good object-oriented design, like extensive coupled classes;
- **Architecture debt:** Refers to problems related to the software architecture, such as violation of modularity;
- **Documentation debt:** Debt related to issues observed in the software documentation;
- **Test debt:** Debt found in testing activities, like planned tests that were not run;
- **Code debt:** Associated with problems found in the source-code, that can make it harder to maintain, usually related to bad coding practices;
- **Build debt:** Refers to issues that can hinder the build task, consuming unnecessary time.

Who is this briefing for?

Software engineering practitioners who want to make decisions about internal quality issues and apply scientific knowledge on managing technical debt.

Where does the information come from?

The information on this briefing comes from evidence collected by the author through a literature review in several publications, included:
- [1] Avgeriou, P. et al. Managing Technical Debt in Software Engineering. In Dagstuhl Reports, 2016;
- [2] Alves, N. S. et al. Identification and management of technical debt: A systematic mapping study. Information and Software Technology, 2016;
- [3] Li, Z. et al. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 2015.

For additional information about the Experimental Software Engineering Group at COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

For additional information about the DELFOS Observatory:

http://www.delfos.cos.ufrj.br

RESEARCH REFERENCE
Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Masters' dissertation, Federal University of Rio de Janeiro, 2018.

Figure 5.2 – EB1 – Technical Debt: Definitions and Concepts

TECHNICAL DEBT MANAGEMENT: ACTIVITIES AND PRACTICES

This evidence briefing reports information on the different activities to manage the technical debt, along with practices obtained through a survey with practitioners and a literature review.

FINDINGS

What are the Technical Debt Management activities?

Through a systematic mapping study, Li et al [1] consolidated the Technical Debt Management (TDM) activities in eight groups, described below:

- **Technical Debt (TD) identification:** Detects TD caused by technical decisions in software, either intentional or unintentional;
- **TD measurement:** Evaluates the cost/benefit relationship of known TD items in software or estimates the overall TD;
- **TD prioritization:** Adopts predefined rules to rank known TD items, to support the decision-making process;
- **TD prevention:** Establishes practices to avoid potential TD from being incurred;
- **TD monitoring:** Observes the evolution of known TD items over time;
- **TD repayment:** Eliminates or reduces the TD impact (principal and interest) in a software system;
- **TD representation/documentation:** Represents and codes TD in a predefined standard, to address the stakeholders' concerns;
- **TD communication:** Disclose the identified TD to the stakeholders.

What are the practices to manage the TD?

The following guidelines or practices were collected on a survey with practitioners from the Brazilian software industry (no participant on the survey answered practices to monitor the TD):

- **TD identification:** manual code inspection, dependency analysis, TD checklist;
- **TD representation/documentation:** TD backlog;
- **TD communication:** Discussion forums, TD meetings;
- **TD prioritization:** Cost/benefit analysis, classification of issues;
- **TD repayment:** Refactoring, redesign, code rewriting;
- **TD prevention:** Coding guidelines or standards, code revision, retrospective meetings, Definition of Done.

The following guidelines or practices were collected from experience reports or case studies with the industry, through a literature review:

13 steps for reducing TD
TDM activity: TD Prevention
TD types covered: Code TD; Architecture TD; Design TD
Source artifact: Any
Type of evidence: Case study
Reference: Krishna, V.; Basu, A. Minimizing Technical Debt: Developer's Viewpoint. International Conference on Software Engineering and Mobile Application Modelling and Development, Chennai, 2012.
4 stages to manage TD in legacy systems
TDM activity: TD Identification; TD Representation/Documentation; TD Prioritization; TD Repayment; TD Prevention
TD types covered: Code TD; Architecture TD; Design TD
Source artifact: Source code
Type of evidence: Case study
Reference: Gupta, R. K. et al. Pragmatic Approach for Managing Technical Debt in Legacy Software Project. 9th India Software Engineering Conference, Goa, 2016.
Approach to control and repay build debt
TDM activity: TD Identification; TD Monitoring; TD Measurement
TD types covered: Build TD
Source artifact: Specifications for building software
Type of evidence: Industry practices at Google
Reference: Morgenthaler, J. D. et al. Searching for build debt: Experiences managing technical debt at Google. 3rd International Workshop on Managing Technical Debt, Piscataway, 2012.
Proactive management of TD by software metrics
TDM activity: TD Measurement
TD types covered: Code TD; Architecture TD; Test TD
Source artifact: Source code
Type of evidence: Industry practices at Ericsson
Reference: Sandberg, A. B.; Staron, M.; Antinyan, V. Towards proactive management of technical debt by software metrics. 14th Symposium on Programming Languages and Software Tools, Tampere, 2015.
Strategies for repaying test debt
TDM activity: TD Repayment
TD types covered: Test TD
Source artifact: Source code
Type of evidence: Case study
Reference: Samarthyam, G.; Muralidharan, M.; Anna, R. K. Understanding Test Debt. Trends in Software Testing, Singapore, 2017
TD board to manage and visualize high level debt
TDM activity: TD Identification; TD Representation/Documentation; TD Monitoring; TD Prioritization; TD Communication
TD types covered: Not specific to a TD type
Source artifact: Source code
Type of evidence: Industry practices at Petrobras
Reference: dos Santos, P. S. M. et al. Visualizing and managing technical debt in agile development: An experience report. International Conference on Agile Software Development, Berlin, 2013.

Who is this briefing for?

Software engineering practitioners who want to make decisions about internal quality issues and apply scientific knowledge on managing technical debt.

Where does the information come from?

The information on this briefing comes from evidence collected by the author through a literature review and a survey with software practitioners in the Brazilian industry. The technical debt management activities were described according to the mapping study listed below:

- [1] Li, Z. et al. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 2015.

For additional information about the Experimental Software Engineering Group at COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

For additional information about the DELFOS Observatory:

http://www.delfos.cos.ufrj.br

RESEARCH REFERENCE

Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Masters' dissertation, Federal University of Rio de Janeiro, 2018.

Silva, V. M.; Jeronimo, H.; Travassos, G. H. ***A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil***. XXI Ibero-American Conference on Software Engineering, Bogotá, 2018.

Figure 5.3 – EB2 – Technical Debt Management: Activities and Practices

TECHNICAL DEBT MANAGEMENT: TOOLS AND STRATEGIES

This evidence briefing reports a list of tools, methodologies, strategies, approaches or frameworks to execute the different Technical Debt (DT) management activities.

Technology	Description	Link	TD type	TDM activity	Source artifact	Evidence	Reference
JSpIRIT	A tool to identify and prioritize technical debt in the form of code smells.	goo.gl/dKnqvp	Code TD Architecture TD Design TD	TD Identification; TD Measurement; TD Prioritization	Source code	CS	[1]
SQALE plugin for SonarQube	A tool to analyze, measure, visualize and prioritize TD based on SQALE quality model.	goo.gl/3oiAys	Code TD Architecture TD	TD Identification; TD Measurement; TD Prioritization; TD Communication	Source code	SP	[2]
SonarQube	An open platform for managing code quality.	goo.gl/6X2KGV	Code TD	TD Identification; TD Measurement; TD Monitoring	Source code	SP	[2]
CheckStyle	A tool to check Java code against coding standards.	goo.gl/iRFBTr	Code TD	TD Identification	Source code	SP	[2]
FindBugs	A tool to identify TD using automatic static analysis.	goo.gl/UFssPz	Code TD	TD Identification	Source code	SP	[2]
JIRA	A tool that allows task monitoring and management.	goo.gl/ga4zMv	Any	TD Representation/Documentation; TD Measurement	N/A	SP	[2]
Trello	A tool that allows task monitoring and management.	goo.gl/iBF6oA	Any	TD Representation/Documentation; TD Communication	N/A	SP	[2]
GitLab	A software repository manager.	goo.gl/kHsxMi	Any	TD Communication	N/A	SP	[2]

CS - Case study

SP - Survey with practitioners

Technology	Description	TD type	TDM activity	Source artifact	Evidence	Reference
CVM-TD	Model to identify TD on code comments.	Code TD Architecture TD Design TD	TD Identification	Source code	IP	[3]
Not named	A decision-support system approach to the modularity debt management.	Design TD	TD Identification; TD Monitoring; TD Measurement; TD Prioritization; TD Repayment	Source code	EE	[4]
Not named	An approach to define manageable levels of technical debt.	Code TD Design TD Test TD	TD Identification; TD Monitoring; TD Measurement; TD Repayment	Source code	IP	[5]
Not named	An approach to quantify TD.	Code TD Design TD	TD Identification; TD Measurement	Source code	IP	[6]
AnaConDebt	A method that aid architects and managers to understand and quantify interest on architecture TD.	Architecture TD	TD Measurement; TD Prioritization	Source code	EE	[7]
Not named	A decision-based approach using a conceptual model of architecture TD.	Architecture TD	Not specific to a TDM activity	Any	EE	[8]
Not named	An identification approach based on architecture decisions and change scenarios.	Architecture TD	TD Identification	Any	EE	[9]
ATD viewpoints	An approach based on set of architecture viewpoints related to architecture TD.	Architecture TD	TD Representation/Documentation	Any	EE	[10]
Not named	A process for TD identification, documentation and prioritization.	Not specific to a TD type	TD Identification; TD Prioritization; TD Representation/Documentation	Any	EE	[11]
Not named	Methodology to help avoiding the acumulation of technical debt.	Code TD Architecture TD Design TD Documentation TD	TD Prevention	Test cases	EE	[12]
Not named	A normative process framework for managing technical debt in commercial software production.	Not specific to a TD type	Not specific to a TDM activity	Any	EE	[13]
SQALE	Method that defines additional indexes and indicators to analyze and understand TD.	Code TD Architecture TD Design TD	TD Identification; TD Measurement; TD Prioritization; TD Repayment; TD Communication	Source code	EE	[14]
TD Template	A framework to support technical debt management.	Not specific to a TD type	Not specific to a TDM activity	Any	EE	[15]
Not named	Visualization approach.	Code TD Design TD Test TD	TD Identification; TD Monitoring	Source code	IP	[16]
Duct taped TD	Visualization technique.	Code TD	TD Representation/Documentation; TD Communication	Source code	IP	[17]
CoBeTDM	A framework to manage and reduce TD.	Code TD Architecture TD	TD Identification; TD Monitoring; TD Prioritization	Any	IP	[18]

EE - Experimentally evaluated

IP - Industry practice

Who is this briefing for?

Software engineering practitioners who want to make decisions about internal quality issues and apply scientific knowledge on managing technical debt.

Where does the information come from?

- [1] Vidal, S. et al. Identifying Architectural Problems through Prioritization of Code Smells. SBCARS, 2016;
- [2] Silva, V. M. et al. A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil. CIbSE, 2018;
- [3] de Freitas Farias, M. A. et al. Investigating the Use of a Contextualized Vocabulary in the Identification of Technical Debt: A Controlled Experiment. ICEIS, 2016;
- [4] Cai, Y. et al. A decision-support system approach to economics-driven modularity evaluation. Economics-Driven Software Architecture, 2014;
- [5] Eisenberg, R. J. A threshold based approach to technical debt. Software Engineering Notes, 2012;
- [6] Nugroho, A. et al. An empirical model of technical debt and interest. Workshop on Managing Technical Debt, 2011;
- [7] Martini, A. et al. An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt. ICSE-C, 2016;
- [8] Li, Z. et al. Architectural debt management in value-oriented architecting. Economics-Driven Software Architecture, 2014;
- [9] Li, Z. et al. Architectural technical debt identification based on architecture decisions and change scenarios. WICSA, 2015;
- [10] Li, Z. et al. Architecture viewpoints for documenting architectural technical debt. Software Quality Assurance, 2016;
- [11] Yli-Huumo, J. et al. Developing Processes to Increase Technical Debt Visibility and Manageability – An Action Research Study in Industry. PROFES, 2016;
- [12] Trumler, W. et al. How "Specification by Example" and Test-Driven Development Help to Avoid Technical Debt. Workshop on Managing Technical Debt, 2016;
- [13] Ramasubbu, N. et al. Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests. IEEE Transactions on Software Engineering, 2017;
- [14] Letouzey, J. L. et al. Managing technical debt with the sqale method. IEEE software, 2012;
- [15] Seaman, C. et al. Measuring and monitoring technical debt. Advances in Computers, 2011;
- [16] Kaiser, M. et al. Selling the Investment to Pay Down Technical Debt: The Code Christmas Tree. AGILE, 2011;
- [17] Chicote, M. Startups and technical debt: managing technical debt with visual thinking. International Workshop on Software Engineering for Startups, 2017;
- [18] Harun, M. F. et al. Towards a technical debt-management framework based on cost-benefit analysis. ICSEA, 2015.

For additional information about the Experimental Software Engineering Group at COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

For additional information about the DELFOS Observatory:

http://www.delfos.cos.ufrj.br

RESEARCH REFERENCE

Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Masters' dissertation, Federal University of Rio de Janeiro, 2018.

Silva, V. M.; Jeronimo, H.; Travassos, G. H. ***A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil***. XXI Ibero-American Conference on Software Engineering, Bogotá, 2018.

Figure 5.4 – EB3 – Technical Debt Management: Tools and Strategies

5.5 Chapter considerations

This chapter presented the concept of EBs and the process for the construction of the briefings containing knowledge acquired in this research. Besides knowledge organization, as Cartaxo et al. (2016) indicate, the EBs are a medium to transfer knowledge from the research findings to the industry so that the practitioners can apply the technologies and information generated in the studies on their day-to-day activities.

Thus, the creation of TD-specific EBs intends to be an instrument to support better dissemination of the TD concepts and associated technologies among software practitioners, potentially improving the overall internal quality of software created in BSOs. Also, although the EBs were partially created using studies related to BSOs, they can be used in other countries.

6 Conclusion

This chapter presents the conclusions of this dissertation. First, some general considerations about the work are detailed, and then the main contributions of this research are discussed, both to the industry and to the academia. Finally, the threats to validity and some future work opportunities are outlined.

6.1 Introduction

This dissertation presented research on TD and technologies available to manage it. First, the central concepts behind the current definition on TD were discussed, including the financial terms incorporated by the metaphor, such as principal, interest, and bankruptcy. The different classifications of TD were also discussed, from the intentional/unintentional debt proposed by McConnell (2007) to the taxonomy of TD proposed initially by Li et al. (2015) and then evolved by Alves et al. (2016).

Observing the gap on TDM research with practitioners from BSOs, Chapter 3 presented the survey conducted with them. The survey gathered information on the current knowledge level of TD and TDM in BSOs, and which are the current technologies adopted by the practitioners to manage the TD. Among other findings, we observed that the general knowledge about TD is considerably low: there are only a small number of practitioners that have heard about TD, and those who have do not manage it. We also observed that the practices adopted to manage the TD are often informal or not mandatory, indicating a low maturity level on practices or strategies to conduct TDM in software projects.

Combining this last finding with conclusions from other studies indicating a lack of available tools to manage the TD (ERNST et al., 2015), we observed a research gap that was addressed in Chapter 4. A *quasi*-systematic literature review was conducted to gather studies that propose new technologies, created specifically for TDM. We collected 77 technologies, supporting all eight TDM activities and covering most of the TD types, as organized by Alves et al. (2016). Our findings were in line with studies that indicated that most technologies are associated with source code and address code-related debt, such as code debt, design debt or architecture debt (LI; AVGERIOU; LIANG, 2015).

Finally, to aggregate knowledge acquired on this dissertation and to simplify the access on the research to software practitioners, evidence briefings were developed and discussed in chapter 5. EBs are a knowledge transfer media proposed by Cartaxo

et al. (2016), in which the findings of primary or secondary researches are condensed in one-page documents. Giving this treatment to the studies provide more direct content to practitioners.

This chapter presents the conclusions of this work, including a revisiting of the research questions from both studies. We also present the main contributions of the research to both academia and industry, discuss the limitations of the studies and evaluate some future work opportunities that raised from the activities conducted in this dissertation. Finally, some final considerations are presented.

6.2 Answering research questions

This section presents a discussion regarding the research questions from both studies conducted in this dissertation.

6.2.1 Survey with practitioners

The survey was planned based on two primary research questions:

6.2.1.1 RQ1: Is there a consensus on the perception of TD among software practitioners?

This research question aimed to assess whether the practitioners from BSOs understand TD as it is defined in academia. First, we asked if the respondent knew about the TD metaphor. Out of the 40 valid answers, only 24 participants affirmed that they already had heard of the term. It led to the indication that the TD definition and applications need better divulgation in the industry.

For those 24 participants that claimed to know what TD is, the survey had a question in which several common issues were listed, and each respondent was asked to select those that could be associated with TD.

Despite observing some alignment between the understandings of TD by the practitioners, some of the misalignments observed must be eliminated to achieve proper use of the metaphor by the practitioners from the industry. The leading example for those misalignments is that half the participants affirmed that issues associated with external quality attributes should be considered TD.

The results of this research question were used as inspiration to develop part of EB1, in which we added some observations regarding the TD definition and some clarification on what should and what should not be considered TD. To enrich the content of EB1, we decided to include how TD can be classified, to help the practitioners understand that its source can be different artifacts other than the source code, and to better detail the TD concept to practitioners that do not know what TD is.

6.2.1.2 RQ2: Do BSOs perceive the TD in their software projects?

The second research question focused on assessing how BSOs perceive the TD, through the practitioner point-of-view. To answer this question, we divided it into the following subquestions:

RQ2.1: Do BSOs manage the TD in their software projects?

This sub-question was answered by asking the practitioners three questions: first, if they perceive the presence of TD in their projects. Only 17 of a total of 40 participants claimed to perceive it. Considering the importance of the topic, this is a relatively low score.

To those participants, we asked if the TD is managed in any way, either through formal practices from the organization or through practices defined by the project manager. Ten of the 17 participants affirmed managing the TD somehow. Considering the total population of 40 participants, this number represents only a small fraction. It could indicate a severe gap in the overall product quality in Brazilian software projects, as this can hinder the projects' evolvability and maintainability.

To address this issue in the EB, we included a section in EB2 explaining the main TDM activities discussed in the technical literature, to illustrate better how the TDM could be laid out in the organizations, either by the project managers or the practitioners themselves.

RQ2.1.1: Which TD management activities are most relevant to software projects?

The goal of this question was to identify, among professionals, which TDM activities are more critical to their practitioners in their projects. Although the results provided insufficient data to include any information on EBs, we could observe that almost half of the participants that answered this section (four out of nine) considered TD prevention as being relevant to a project. Other TDM activities considered relevant were TD identification and TD prioritization. This information, associated with results from the qSLR presented in chapter 4 can supply researchers with potential work opportunities, further discussed in section 6.5.

RQ2.1.2: Which technologies and strategies are adopted for each TDM activity?

For each TDM activity, the survey raised questions regarding which practices and technologies are used by practitioners to support the activity. Despite the existence of other studies discussing this matter, it was essential to assess if practitioners from BSOs also adopted the same practices and technologies that are adopted worldwide.

The results presented the 15 practices and seven tools that are adopted by practitioners from BSOs. Those technologies were included in EB2 (practices) and EB3 (tools), organized by TDM activities that the technology supports and/or TD types covered by the application of the technology. It will provide information to the practitioner, so s/he can assess the necessities of the organization and further evaluate more appropriate technologies to attend those necessities.

6.2.2 *quasi*-Systematic Literature Review

This study had one central research question, followed by five sub-questions:

- RQ1: Which software technologies have been proposed in the technical debt literature to support the technical debt management in the context of software projects?
    - RQ1.1: Which TDM activities are covered by those technologies?
    - RQ1.2: Which TD types are supported by those technologies?
    - RQ1.3: What is the software context in which those technologies should be applied?
    - RQ1.4: Which other technologies, from the Software Engineering context or other domains, are necessary to apply those technologies?
    - RQ1.5: Are there evidence on the use or applicability of those technologies?

Unlike the previous study, which provided information on the understanding and perception of the practitioners regarding TD and TDM, this research focused exclusively on TDM-specific technologies, resulting in 77 technologies supporting TDM. 23 of these were included in EB2 (practices) and EB3 (tools and other strategies, methodologies or frameworks).

We decided not to include all technologies since the target population for EBs is practitioners in software organizations. This decision was made as we cannot provide information on technologies that are not available (e.g., tools that do not have a download link) or technologies that were not adequately evaluated in real-life scenarios (e.g., toy examples or simulations).

6.3 Contributions

It is possible to analyze the contributions of this research under two perspectives:

6.3.1 To the academic community

Our survey research on TD and TDM among practitioners from BSOs generated information about the current knowledge level of TD in Brazil and indicated that practitioners in Brazil often do not know about TD. Therefore, our research supplied indications that there is a need to conduct studies on how to disseminate this knowledge in the Brazilian industry. It could lead, for example, to action researches on BSOs to observe the inclusion of TDM processes and activities on the software development.

Another academic contribution of the survey was a research package containing the survey plan and questionnaires in English and Portuguese, to enable the replication of the study in other contexts. This material can be used as a means to compare the general maturity level on TDM among different countries, for example.

The literature review on TDM technologies provided information on which technologies were researched and developed specifically to manage the TD. One of the leading academic contributions of this research is the indication that almost half of the studies did not evaluate the technologies, presenting at most a proof-of-concept. It calls for attention, as those technologies should be further evaluated before they could be applied in real projects by practitioners in the industry.

Another contribution is the indication that there are some research areas on TDM that could be over-researched, like technologies covering code debt. However, we also identified vast research opportunities on TD types that are barely addressed in the technical literature, as defect debt or documentation debt. It could provide several research opportunities for future academic studies.

Finally, the evidence briefings on TD and TDM, although directed to the practitioners in the industry, could serve as a means to disseminate the culture of synthesizing the academic research on TD and TDM, to provide easy access of state-of-the-art technologies and concepts to software practitioners, improving the penetration of these studies in the industry and supporting a better integration between academia and industry.

6.3.2 To the industry

The survey research served as evidence for practitioners that TD concepts must be better understood and disseminated so that they can be appropriately used in BSOs. The research also indicated that the perception of TD in the industry is still low and that BSOs have, in general, a low maturity level on TDM.

Moreover, the survey presented a list of 22 technologies that are currently used by software practitioners to manage the TD in software projects. This list was

complemented by technologies collected in the *quasi*-systematic literature review, which listed 77 TDM technologies.

Both lists, along with the TD definitions and concepts obtained in the initial literature review, were aggregated to form the evidence briefings, which primary purpose is to transfer knowledge from academic researches to the industry. These EBs were designed to be used to help disseminate the TD concept to software practitioners, potentially increasing the internal quality of software projects in BSOs.

6.4 Limitations

As with any research work, this dissertation also has limitations that were discussed throughout the chapters. They present threats to validity not only in the studies but also to the dissertation itself.

6.4.1 Limitations on the studies

For the survey research, the main limitation is the generalization of the results, as their confidence level might be low. Therefore, the findings of the survey might not be applied to the entire population. The possibility of the participants misunderstanding some of the questions from the survey also constitutes a potential threat to validity. Finally, despite being revised thoroughly by two researchers during the design phase and having two pilot trials to assess the quality among practitioners, the survey could be released with some level of bias.

The results of the qSLR also have some limitations, including the identification of primary studies, the researchers' bias on the study selection, the quality assessment, and the data synthesis; and the inaccuracy of the data extracted.

The limitation on the identification of primary studies was mitigated through a control list created from previous secondary studies, and the application of the forward and backward snowballing technique to obtain studies that did not return on the search engine. The limitation on the researchers' bias was mitigated through a revision of the information by both researchers of the qSLR, and the registration and discussion of the doubts rose during the study selection. Similarly, the data inaccuracy on the extraction of the selected studies was also mitigated through a revision of the content by both researchers involved in the literature review.

Finally, the evidence briefings also have some limitations, most of them associated with the previous studies. For instance, the generalization limitation from the survey research also applies here, as it is not possible to ensure that the list of technologies obtained from the survey is applied to all BSOs. Similarly, the limitation on the identification of primary studies from the qSLR is also observable in the EBs, since

there could exist studies that were not covered in the review, despite our efforts in reducing this threat. Notwithstanding, the EBs attend their objective as an instrument to aid practitioners to start applying the concepts of TD and using the TDM technologies in their software projects.

6.4.2 Limitations on the dissertation

It is also worth to discuss not the methodological limitations of the studies conducted in this dissertation, but also the limitations of the dissertation itself, highlighting some possible evolution aspects that could be taken to improve the studies.

First, regarding the survey, we observed that a considerable percentage of the participants did not know what TD is. As a consequence, those participants did not answer any other relevant questions. This could have been avoided if the survey had a brief TD definition to explain to those unaware participants so that they could have answered the following questions. This strategy was used in Holvitie et al. (2018), improving the response rate of the survey.

Moreover, an improvement on the response rate could have been achieved if the welcome message and the survey disclosure were designed not to detail the TD topic, as most people that did not know anything about it might have lost interest on the survey. If the survey were released under the “internal quality” general topic, maybe we could have had a better response rate.

Concerning the literature review, due to schedule issues, we did not include some aspects that could have increased the research scope. For example, conducting a multivocal literature review, including blog posts and other developers’ familiar sources (such as *StackOverflow*) could have brought a more extensive range of TDM technologies. Also, we did not evaluate follow-up studies on the technologies identified by the review. It could have shown an improvement in the overall quality assessment of the studies, as probably most first papers on technology do not evaluate it.

Finally, regarding the EBs, a certain improvement point is regarding its evaluation among practitioners. Although the effectiveness of the EBs in the industry was already assessed by Cartaxo et al. (2016), we could have investigated the application of our EBs in BSOs. However, we had time constraints that would turn this evaluation unfeasible, being then described as a future work opportunity (see the following section for more details).

6.5 Future work opportunities

Considering the research conducted in this work, and analyzing the conclusions presented in each chapter, it is possible to suggest some future work opportunities, discussed in this subsection.

To ensure the applicability of EBs into the BSOs, and to enable potential evolutions of the material, research to evaluate the briefings among software practitioners in BSOs could result in considerable contributions for the TDM scenario in the Brazilian industry. Similarly, EBs and the results from the survey indicate a possible future work on case studies or action researches, to study how the organizations are including TDM practices and technologies in their software development process.

From the qSLR, we identify some opportunities related to the low coverage of technologies regarding some TDM activities, like TD communication or TD repayment. There is a research gap on developing new technologies to attend these activities. Likewise, there is also a research gap in treating specific types of TD, such as build debt and service debt.

We also observed several technologies that have very little to no evaluation. Therefore, it is possible to plan studies to evaluate those technologies better, so they can be applied in the industry.

Finally, one last future research opportunity is regarding the possible integration of different technologies that could be complementary to each other, developing solutions that attend a more extensive range of demands from TDM in general.

6.6 Final considerations

This chapter summarized the research conducted throughout this dissertation, including the contributions to the industry and academic community, as well as a set of possible future work opportunities to improve the general knowledge on TD and TDM among the software practitioners.

We observed that, despite TD being of critical importance to good software health, only a few practitioners from BSOs manage it. It is not possible to conclude, to the best of our knowledge, if TD is being tacitly managed in BSOs, without referencing the concepts described in this dissertation. Regardless, it is essential to develop research and solutions to reduce this gap between industry and academia, aiming at better management of TD and, consequently, at software products with better quality, maintainable and evolvable.

References

ALVES, N. S. R. et al. "Identification and management of technical debt: A systematic mapping study." **Information and Software Technology**, v. 70, p. 100–121, 2016.

AMPATZOGLOU, A. et al. "The financial aspect of managing technical debt: A systematic literature review." **Information and Software Technology**, v. 64, p. 52-73, 2015.

AMPATZOGLOU, A. et al. "The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study." In: **8th International Workshop on Managing Technical Debt**. IEEE, 2016. p. 9-16.

ASSUNCAO, T. R. DE et al. "Technical Debt Management in the Brazilian Federal Administration." In: **6th Brazilian Workshop on Agile Methods**. IEEE, 2015, p. 6-9.

AVGERIOU, P. et al. "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)". In: **Dagstuhl Reports**. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.

BROWN, N. et al. "Managing technical debt in software-reliant systems." In: **Workshop on Future of software engineering research**. ACM, 2010. p. 47-52.

CARTAXO, B. et al. "Evidence Briefings: Towards a Medium to Transfer Knowledge from Systematic Reviews to Practitioners." In: **International Symposium on Empirical Software Engineering and Measurement**. ACM, 2016. p. 57.

CODABUX, Z. et al. "An empirical assessment of technical debt practices in the industry." **Journal of Software: Evolution and Process**, v. 29, n. 10, p. e1894, 2017.

CUNNINGHAM, W. et al. "The WyCash portfolio management system." **ACM SIGPLAN OOPS Messenger**, v. 4, n. 2, p. 29–30, 1 Apr. 1993.

CURTIS, B.; SAPPIDI, J.; SZYNKARSKI, A. "Estimating the Principal of an Application's Technical Debt." **IEEE Software**, v. 29, n. 6, p. 34–42, 2012.

DAVIS, J. D.; ANDERSEN, T. J. "Surviving the economic downturn." In: **Agile Conference**. IEEE, 2009. p. 245-250.

DE ALMEIDA BIOLCHINI, J. C. et al. "Scientific research ontology to support systematic review in software engineering." **Advanced Engineering Informatics**, v. 21, n. 2, p. 133–151, Apr. 2007.

DOS SANTOS, P. S. et al. "Visualizing and Managing Technical Debt in Agile Development: An Experience Report." In: **International Conference on Agile Software Development**. Springer, Berlin, Heidelberg, 2013. p. 121–134.

EISENBERG, R. J. "A threshold-based approach to technical debt." **ACM SIGSOFT Software Engineering Notes**, v. 37, n. 2, p. 1, 2012.

ERNST, N. A. et al. "Measure it? Manage it? Ignore it? software practitioners and technical debt". In: **10th Joint Meeting on Foundations of Software Engineering**. ACM, 2015. p. 50-60.

FOWLER, M. "Technical Debt Quadrant." Available in: <https://martinfowler.com/bliki/TechnicalDebtQuadrant.html>. Access in: July 1st, 2018.

GARTNER INC. "Gartner Estimates Global 'IT Debt' to Be $500 Billion This Year, with Potential to Grow to $1 Trillion by 2015". Available in: <https://www.gartner.com/newsroom/id/1439513>. Access in: July 1st, 2018.

GUAMAN, D. et al. "SonarQube as a tool to identify software metrics and technical debt in the source code through static analysis." In: **7th International Workshop on Computer Science and Engineering**. 2017. p. 171-175.

GUAMAN, D. et al. "Uso de SQALE y herramientas para análisis e identificación de deuda técnica de código a través de análisis estático". In: **Iberian Conference on Information Systems and Technologies**. IEEE, 2017.

GUO, Y.; SPÍNOLA, R. O.; SEAMAN, C. "Exploring the costs of technical debt management – a case study." **Empirical Software Engineering**, v. 21, n. 1, p. 159–182, 2016.

HOLVITIE, J. "Software implementation knowledge management with technical debt and network analysis." In: **International Conference on Research Challenges in Information Science**. IEEE, 2014. p.1-6.

HOLVITIE, J. et al. "Technical debt and agile software development practices and processes: An industry practitioner survey." **Information and Software Technology**, v. 96, p. 141–160, 2018.

HOLVITIE, J.; LEPPANEN, V. "DebtFlag: Technical debt management with a development environment integrated tool". In: **4th International Workshop on Managing Technical Debt**. IEEE, 2013. p. 20–27.

KITCHENHAM, B. **Procedures for Performing Systematic Reviews**. Keele, UK: [s.n.].

KLINGER, T. et al. "An enterprise perspective on technical debt". In: 2nd Workshop on Managing technical debt. ACM, 2011. p. 35-38.

KRISHNA, V. "Software Engineering Practices for Minimizing Technical Debt." In: **International Conference on Software Engineering Research and Practice**. 2013.

KRISHNA, V.; BASU, A. "Minimizing technical debt: developer's viewpoint." In:

International Conference on Software Engineering and Mobile Application Modelling and Development. 2012.

KRUCHTEN, P.; NORD, R. L.; OZKAYA, I. "Technical debt: From metaphor to theory and practice." **IEEE Software**, v. 29, n. 6, p. 18-21, 2012.

KTATA, O.; LÉVESQUE, G. "Designing and implementing a measurement program for Scrum teams." In: **Third C* Conference on Computer Science and Software Engineering**. ACM, 2010. p. 101-107.

LETOUZEY, J.-L. "The SQALE Method for Evaluating Technical Debt." In: **Third International Workshop on Managing Technical Debt**. IEEE, 2012. p. 31-36.

LI, Z.; AVGERIOU, P.; LIANG, P. "A systematic mapping study of technical debt and its management." **Journal of Systems and Software**, v. 101, p. 193–220, 2015.

LIM, E.; TAKSANDE, N.; SEAMAN, C. "A balancing act: What software practitioners have to say about technical debt." **IEEE Software**, v. 29, n. 6, p. 22-27, 2012.

LINÂKER, J. et al. **Guidelines for conducting surveys in software engineering**. [s.l: s.n.].

MCCONNELL, S. "Technical Debt - 10x Software Development". Available in: <http://www.construx.com/10x_Software_Development/Technical_Debt/>. Access in: January 23rd, 2018.

OLIVEIRA, F.; GOLDMAN, A.; SANTOS, V. "Managing Technical Debt in Software Projects Using Scrum: An Action Research." In: **Agile Conference**. IEEE, 2015.

PAI, M. et al. "Systematic reviews and meta-analyses: an illustrated, step-by-step guide." **The National medical journal of India**, v. 17 2, p. 86–95, 2004.

PFLEEGER, S. .; L., S. "Understanding and improving technology transfer in software engineering." **Journal of Systems and Software**, v. 47, n. 2–3, p. 111–124, Jul. 1999.

PRIKLADNICKI, R. et al. "Distributed software development: Practices and challenges in different business strategies of offshoring and onshoring." In: **International Conference on Global Software Engineering**. IEEE, 2007.

RIBEIRO, L. F. et al. "Decision Criteria for the Payment of Technical Debt in Software Projects : A Systematic Mapping Study." In **18th International Conference on Enterprise Information Systems**. Science and Technology Publications, 2016.

RIBEIRO, L. F. et al. "A strategy based on multiple decision criteria to support technical debt management." In: **43rd Euromicro Conference on Software Engineering and Advanced Applications**. IEEE, 2017.

RIBEIRO, L. F.; SPÍNOLA, O. "Um Survey sobre a Pertinência e Relevância de Critérios de Decisão para Apoiar o Gerenciamento de Itens de Dívida Técnica". In: **XV Simpósio Brasileiro de Qualidade de Software**, p. 256–270, 2016.

ROCHA, J. C.; ZAPALOWSKI, V.; NUNES, I. "Understanding Technical Debt at the Code Level from the Perspective of Software Developers." In: **31st Brazilian Symposium on Software Engineering**. ACM, 2017.

SAPPIDI, J.; CURTIS, B.; SZYNKARSKI, A. **The CRASH Report - 2011/12**. [s.l: s.n.].

SEAMAN, C.; GUO, Y. "Measuring and Monitoring Technical Debt." **Advances in Computers**, v. 82, p. 25–46, 1 Jan. 2011.

SILVA, V. M.; JERONIMO, H.; TRAVASSOS, G. H. "A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil." In: **Ibero-American Conference on Software Engineering**. Bogotá: 2018.

SILVA, V. M.; JERONIMO, H.; TRAVASSOS, G. H. "Technical Debt Management in Brazilian Software Organizations: A Need, an Expectation, or a Fact?" In: **Brazilian Symposium on Software Quality**. Curitiba: 2018 (accepted paper).

SPÍNOLA, R. O. et al. "Investigating technical debt folklore: Shedding some light on technical debt opinion." In: **4th International Workshop on Managing Technical Debt**. IEEE, 2013.

TOM, E.; AURUM, A.; VIDGEN, R. "An exploration of technical debt." **Journal of Systems and Software**, v. 86, n. 6, p. 1498–1516, Jun. 2013.

TONIN, G. S., 2018, **Technical Debt Management in the Context of Agile Methods in Software Development**. Masters' dissertation, University of São Paulo, São Paulo, SP, Brazil.

TRAVASSOS, G. H. et al. "An environment to support large-scale experimentation in software engineering." In: **International Conference on Engineering of Complex Computer Systems**. IEEE, 2008.

VAN SOLINGEN, R. et al. "Goal Question Metric (GQM) Approach." In: **Encyclopedia of Software Engineering**. Hoboken, NJ, USA: John Wiley & Sons, Inc., 2002.

WOHLIN, C. "Guidelines for snowballing in systematic literature studies and replication in software engineering." In: **18th International Conference on Evaluation and Assessment in Software Engineering**. ACM, 2014.

YLI-HUUMO, J. et al. "Developing processes to increase technical debt visibility and manageability – An action research study in the industry." In: **Lecture Notes in Computer Science**. Springer, 2016.

YLI-HUUMO, J.; MAGLYAS, A.; SMOLANDER, K. "How do software development teams manage technical debt? – An empirical study". **Journal of Systems and Software**, v. 120, p. 195–218, 1 out. 2016.

ZAZWORKA, N. et al. "A case study on effectively identifying technical debt." In: **17th International Conference on Evaluation and Assessment in Software Engineering**. ACM, 2013.

ZAZWORKA, N. et al. "Comparing four approaches for technical debt identification." **Software Quality Journal**, v. 22, n. 3, p. 403–426, 3 set. 2014.

Appendix A – Survey Questionnaire

This appendix presents the final questionnaire adopted on the survey, in the original version, and in English.

A.1 Introduction

In this appendix, we present the survey questionnaire in two formats: the original version in Portuguese, used to apply the study in Brazil, and an equivalent version in English, to simplify the replication process to other countries.

For this appendix, all questions are provided underlined. Following the question, a text in *italic* indicates in which scale that question was designed. The answers follow a specific format, described from now on:

- Questions with only one possible answer: in this case, the answers are preceded by parentheses "( )." Only one answer is possible for this type of question;
- Questions with multiple answers: this type of question has answers preceded by brackets "[ ]." The participant can select any number of answers for this type of question;
- Answers in which the participant can insert text or numbers: curly brackets will follow this type of answer, indicating the values accepted, e.g., "{ string }" or "{ numbers between 1 and 8 }."

Any observations included throughout the questionnaire are indicated by the abbreviation "**obs.**", and should not be displayed to the participant. A horizontal line between questions indicates that we included a page break in the questionnaire, to facilitate the user interaction or to avoid bias in the answers.

A.2 Original version

A.2.1 Apresentação

Este trabalho visa observar como o gerenciamento da dívida técnica vem sendo tratado no cenário das organizações de software brasileiras. O objetivo é caracterizar a perspectiva da indústria quanto a identificação, medição, priorização e outras atividades de gerenciamento da dívida técnica em projetos de software e comparar com as recentes pesquisas sobre o tema.

Os resultados desta pesquisa são de relevância para a indústria de software brasileira, já que busca identificar estratégias e tecnologias de gerenciamento da dívi-

da técnica que possam ser aprimoradas ou empregadas no contexto do desenvolvimento de software no Brasil.

A participação do profissional de software neste *survey* é voluntária e anônima. Entretanto, é muito importante e relevante a sua participação! Quanto maior a participação dos profissionais da prática, mais significativos serão os resultados e as contribuições deste trabalho para a comunidade de software.

O questionário é composto por dois conjuntos de perguntas: uma caracterização do participante, da sua organização e do projeto em que ele(a) está inserido(a), que possui tempo estimado de 5 minutos; e um questionário técnico, envolvendo perguntas relacionadas à dívida técnica e ao gerenciamento da mesma. Para a segunda parte, o tempo esperado de resposta depende do arranjo de trabalho do profissional e varia de 5 a 20 minutos.

Este *survey* está inserido nas atividades de pesquisa e desenvolvimento relacionadas a dissertação de mestrado de Victor Machado da Silva, na linha de Engenharia de Software do Programa de Engenharia de Sistemas e Computação – PESC (COPPE/UFRJ), sob orientação do prof. Guilherme Horta Travassos.

Esta pesquisa adota os princípios éticos e científicos que norteiam a pesquisa científica em engenharia de software experimental. Desta forma, dados pessoais e sensíveis não são solicitados, a participação é totalmente voluntária e anônima. Todos os resultados serão apresentados de forma agregada, sem a possibilidade de identificação do respondente.

Caso tenha dúvidas ou sugestões, por favor entre em contato através do e-mail victorms@cos.ufrj.br.

A.2.2 Caracterização do participante

Esta seção tem por objetivo obter algumas informações profissionais sobre você.

Questão 1: Qual a sua função no projeto de software atual ou mais recente que participou? Por favor escolha as opções que se aplicam: *Escala nominal*

- [ ] Gerente de Projeto
- [ ] Gerente de Linha de Produto
- [ ] Líder de Equipe
- [ ] Arquiteto de Software
- [ ] Engenheiro de Teste de Software
- [ ] Programador
- [ ] Designer de Interface de Usuário
- [ ] Analista de Requisitos

- [ ] Pesquisador
- [ ] Outro { string }

Questão 2: Qual a sua formação acadêmica? *Escala nominal*

- ( ) Técnico/Ensino Médio completo
- ( ) Graduação incompleta
- ( ) Graduação completa
- ( ) Pós-graduação completa (especialização)
- ( ) Pós-graduação completa (mestrado)
- ( ) Pós-graduação completa (doutorado)

Questão 3: Qual a sua experiência, em anos de trabalho, com projetos de software? Apenas números podem ser usados nesse campo. *Escala razão*

- { números inteiros positivos }

A.2.3 Caracterização da organização

Esta seção tem por objetivo obter algumas informações sobre a sua organização. Responda sobre a organização mais recente em que você trabalhou, ou está atualmente trabalhando.

Questão 4: Qual o setor de atuação da sua organização? *Escala nominal*

- ( ) Tecnologia da Informação
- ( ) Telecomunicações
- ( ) Médico/Biológico/Farmacêutico
- ( ) Finanças (Seguros, Bancos, etc.)
- ( ) Governo
- ( ) Consultoria
- ( ) Educação
- ( ) Outros { string }

Questão 5: Qual o tamanho da sua organização, em número de funcionários? *Escala intervalar*

- ( ) Menos de 10 funcionários
- ( ) Entre 10 e 49 funcionários
- ( ) Entre 50 e 99 funcionários
- ( ) Mais de 100 funcionários
- ( ) Não sei

Questão 6: A sua organização possui alguma avaliação de maturidade em processos de software (MPS.BR, CMMI, ...) vigente? *Escala nominal*

- [ ] Possui avaliação MPS.BR

- [ ] Possui avaliação CMMI
- [ ] Possui outra avaliação
- [ ] Não sei
- [ ] Não possui nenhuma avaliação

Obs.: Caso o participante responda “não sei” ou “não possui nenhuma avaliação”, as três primeiras respostas devem ficar impossibilitadas de ser marcadas. As questões 7, 8 e 9 só serão exibidas para o participante caso tenham sido marcadas as respostas “possui avaliação MPS.BR”, “possui avaliação CMMI” ou “possui outra avaliação”, respectivamente.

Questão 7: Qual o nível MPS.BR que a sua organização possui? Responda a essa pergunta apenas se tiver respondido “possui avaliação MPS.BR” na questão 6. *Escala ordinal*

- ( ) A
- ( ) B
- ( ) C
- ( ) D
- ( ) E
- ( ) F
- ( ) G

Questão 8: Qual o nível CMMI que a sua organização possui? Responda a essa pergunta apenas se tiver respondido “possui avaliação CMMI” na questão 6. *Escala ordinal*

- ( ) 1
- ( ) 2
- ( ) 3
- ( ) 4
- ( ) 5

Questão 9: Caso tenha respondido que a sua organização possui outra avaliação que não MPS.BR ou CMMI, cite o nome da avaliação e o nível em que a organização foi avaliação (se houver). *Escala nominal*

- { string }

A.2.4 Caracterização do projeto

Esta seção tem por objetivo obter algumas informações sobre o projeto que será usado como referência para todas as questões deste survey. Quando a pergunta

tratar do seu “projeto mais recente”, responda sobre o projeto mais recente que trabalhou, ou que está atualmente trabalhando.

Questão 10: Qual o domínio de problema se aplica ao seu projeto de software atual ou mais recente? *Escala nominal*

- ( ) Telecomunicações
- ( ) Educação
- ( ) Governo
- ( ) Imagem
- ( ) Varejo, distribuição e transporte
- ( ) Finanças
- ( ) Sistemas médicos
- ( ) Defesa
- ( ) Farmacêuticos
- ( ) Outros { string }

Questão 11: Qual é o modelo de ciclo de vida de desenvolvimento do seu projeto de software atual ou mais recente? *Escala nominal*

- ( ) Cascata
- ( ) Incremental
- ( ) Espiral
- ( ) Ágil
- ( ) Outros { string }

A.2.5 Entendimento e percepção da dívida técnica

Esta parte do questionário busca obter o entendimento geral dos participantes com relação à definição da dívida técnica e suas características.

Questão 12: Você sabe do que se trata o termo “dívida técnica”? *Escala nominal*

- ( ) Sim
- ( ) Não
- ( ) Sim, mas conheço por outro nome (cite qual) { string }

Obs.: O restante do questionário só será exibido para o participante caso a resposta para a questão 12 seja “sim” ou “sim, mas conheço por outro nome”.

Questão 13: Na sua opinião, quais itens abaixo podem ser associados ao conceito da dívida técnica? Marque todos os itens que considerar pertinente. *Escala nominal*

- [ ] Defeitos

- [ ] Funcionalidades requeridas, porém não implementadas
- [ ] Falta de processos de apoio à realização das atividades do projeto
- [ ] Tarefas planejadas, porém não realizadas ou não finalizadas (ex.: especificação de requisitos, modelos, planos de testes, execução de testes, etc.)
- [ ] Questões de qualidade de código triviais, que não violem as regras de código
- [ ] Questões associadas à baixa qualidade externa, como usabilidade e eficiência
- [ ] Questões associadas à baixa qualidade interna, como manutenibilidade e reusabilidade
- [ ] Problemas arquiteturais (como violação de modularidade)
- [ ] "Atalhos" tomados durante o design
- [ ] Código mal escrito e que viole regras de código
- [ ] Presença de defeitos conhecidos e não corrigidos
- [ ] "Code smells"

Questão 14: No projeto de software atual ou mais recente que você participou, foi percebida a existência de algum problema que foi associado ao conceito de dívida técnica? *Escala nominal*

- ( ) Sim
- ( ) Não

Obs.: O restante do questionário só será exibido para o participante caso a resposta para a questão 14 seja "sim".

Questão 15: Na sua organização são executadas atividades que gerenciam a dívida técnica, mesmo que parcialmente? *Escala nominal*

- ( ) Sim
- ( ) Não

Questão 16: No projeto mais recente em que você participou, você (ou o gerente do projeto) adota atividades adicionais aos definidos pela organização para gerencia a dívida técnica? *Escala nominal*

- ( ) Sim
- ( ) Não

Obs.: O restante do questionário só será exibido para o participante caso a resposta de pelo menos uma das questões 15 e 16 seja "sim".

A.2.6 Gerenciamento da dívida técnica

Este grupo de questões busca identificar, de forma geral, a adoção e relevância de práticas de gerenciamento da dívida técnica nos projetos da indústria de software brasileira. Por “gerenciamento da dívida técnica”, entenda como todas as atividades desenvolvidas com o propósito de organizar, monitorar e controlar a dívida técnica e seus impactos no projeto de software.

Para todas as questões, responda usando como base o projeto mais recente em que atuou (ou que está atuando).

Questão 17: Marque se o seu projeto mais recente aplica uma ou mais das atividades abaixo, seja através de uma estratégia definida ou através de práticas informais. *Escala nominal*

- [ ] Identificação da dívida técnica – visa detectar itens de dívida causados por decisões técnicas intencionais ou não-intencionais nos artefatos de software
- [ ] Documentação/representação da dívida técnica – busca registrar os itens de dívida técnica a fim de um adequado gerenciamento
- [ ] Comunicação da dívida técnica – tem por objetivo tornar visíveis os itens de dívida técnica identificados a todas as partes interessadas
- [ ] Medição da dívida técnica – visa quantificar a dívida técnica já identificada em uma determinada unidade de medida (tempo, valor monetário, etc.) e também avaliar a relação custo/benefício de dívidas técnicas conhecidas em um sistema de software
- [ ] Priorização da dívida técnica – ordenação dos itens de dívida técnica identificados de acordo com regras pré-definidas para apoiar a tomada de decisão de quais itens devem ser pagos primeiramente
- [ ] Pagamento da dívida técnica – trata da solução ou mitigação da dívida técnica em um sistema de software
- [ ] Monitoramento da dívida técnica – atividade de observação das mudanças das relações custo/benefício de dívidas técnicas não solucionadas ao longo do tempo
- [ ] Prevenção da dívida técnica – busca prevenir a ocorrência de potenciais itens de dívida técnica
- [ ] Outra(s) atividade(s)

Obs.: A questão 18 só será exibida para o participante caso a resposta “outra(s) atividade(s)” tenha sido marcada.

Questão 18: Cite uma das atividades de gerenciamento da dívida técnica que seu projeto mais recente atua e não foi citado nas questões anteriores. *Escala nominal*

- { string }

Obs.: Nas questões 19 e 20 só deverão ser exibidas as atividades de gerenciamento que foram selecionadas pelo participante na questão 17.

Questão 19: Marque, para cada uma das atividades listadas abaixo quais são os responsáveis pela sua execução, no seu projeto mais recente. *Escala nominal*

	Gerente do projeto	Líder da equipe	Arquit. de software	Equipe de desenv.	Nenhum dos ant.
Identificação da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Documentação/representação da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Comunicação da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Medição da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Priorização da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Pagamento da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Monitoramento da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Prevenção da dívida técnica	[ ]	[ ]	[ ]	[ ]	[ ]
Outra atividade citada anteriormente	[ ]	[ ]	[ ]	[ ]	[ ]

Questão 20: Caso o seu projeto mais recente possua outros responsáveis para as atividades listadas, além das funções já apresentadas na questão anterior, preencha os campos abaixo. Se não houver nenhum outro responsável, digite “Nada a declarar”. *Escala nominal*

- Identificação da dívida técnica { string }
- Documentação/representação da dívida técnica { string }
- Comunicação da dívida técnica { string }
- Medição da dívida técnica { string }
- Priorização da dívida técnica { string }

- Pagamento da dívida técnica { string }
- Monitoramento da dívida técnica { string }
- Prevenção da dívida técnica { string }
- Outra atividade citada anteriormente { string }

Questão 21: Ordene as atividades listadas abaixo de acordo com a sua opinião com relação à importância para o seu projeto mais recente. Considere que "1" é a atividade menos importante e "8" a mais importante. Não é possível marcar duas atividades na mesma ordem. Ou seja, todas elas devem possuir graus de importância distintos. Escala nominal

- Identificação da dívida técnica { número entre 1 e 8 }
- Documentação/representação da dívida técnica { número entre 1 e 8 }
- Comunicação da dívida técnica { número entre 1 e 8 }
- Medição da dívida técnica { número entre 1 e 8 }
- Priorização da dívida técnica { número entre 1 e 8 }
- Pagamento da dívida técnica { número entre 1 e 8 }
- Monitoramento da dívida técnica { número entre 1 e 8 }
- Prevenção da dívida técnica { número entre 1 e 8 }

Obs.: As próximas seções só serão exibidas para os participantes que tiverem marcado as respectivas atividades na questão 17. Por exemplo, se o participante respondeu que adota as atividades Identificação da dívida técnica e Monitoramento da dívida técnica, apenas essas duas atividades serão habilitadas para resposta.

A.2.7 Identificação da dívida técnica

A atividade de identificação da dívida técnica visa detectar itens de dívida causados por decisões técnicas intencionais ou não-intencionais. Baseado nesta definição, responda as seguintes questões, referentes ao seu projeto mais recente.

Questão 22: Existe uma prática formalizada para a identificação da dívida técnica? *Escala nominal*

- ( ) Sim, o projeto possui uma prática formalizada para identificação da dívida técnica
- ( ) Não, a identificação da dívida é feita apenas informalmente

Obs.: A questão 23 só será exibida para o participante caso a resposta da questão 22 tenha sido "sim".

Questão 23: A estratégia de identificação da dívida técnica deve ser aplicada por todos os envolvidos nesta atividade? *Escala nominal*

- ( ) Sim, a aplicação da estratégia de identificação da dívida técnica é obrigatória para todos os envolvidos na atividade
- ( ) Não, a aplicação da estratégia de identificação da dívida técnica é opcional para os envolvidos na atividade

Questão 24: Em que momento do projeto de software a dívida técnica é identificada? *Escala nominal*

- ( ) Não ocorre em um momento definido, identificamos a dívida técnica apenas esporadicamente ao longo do projeto
- ( ) Não ocorre em um momento definido, identificamos a dívida técnica sempre que percebemos algum problema
- ( ) Sempre identificamos a dívida técnica ao final de cada iteração/sprint
- ( ) A identificação da dívida técnica é contínua, ou seja, ocorre ao longo de todo o processo de desenvolvimento

Questão 25: Marque na lista abaixo todas as ferramentas ou técnicas que são usadas para a identificação da dívida técnica. *Escala nominal*

- [ ] Inspeção manual do código
- [ ] Análise de dependências
- [ ] Checklist
- [ ] SonarQube/SQALE
- [ ] CheckStyle
- [ ] FindBugs
- [ ] CodeVizard
- [ ] CLIO
- [ ] Outras práticas/ferramentas (cite quais): { string }

Questão 26: A dívida técnica é classificada de alguma forma? *Escala nominal*

- ( ) Sim, possuímos critérios de classificação para a melhor organização dos itens de dívida técnica identificados
- ( ) Não, a dívida técnica é apenas identificada e não há uma classificação específica entre os itens

Obs.: A questão 27 só deverá ser exibida para o participante caso a resposta da questão 26 tenha sido “sim”.

Questão 27: Como a dívida técnica é classificada? Marque todas as opções relevantes. *Escala nominal*

- [ ] Tipo da dívida técnica (dívida de arquitetura, dívida de código, etc.)
- [ ] Dimensão da dívida técnica (intencional/não intencional, estratégica/tática, etc.)

- [ ] Artefato de origem (código, diagramas, requisitos, etc.)
- [ ] Classificação própria (se possível, descrever resumidamente) { string }

Obs.: A questão 28 só deverá ser exibida para o participante caso a resposta da questão 27 tenha sido “tipo da dívida técnica”.

Questão 28: Se a dívida técnica é classificada de acordo com o seu tipo, marque quais tipos de dívida são identificados:

- [ ] Dívida de projeto
- [ ] Dívida de arquitetura
- [ ] Dívida de documentação
- [ ] Dívida de teste
- [ ] Dívida de código
- [ ] Dívida de defeito
- [ ] Dívida de requisitos
- [ ] Dívida de infraestrutura
- [ ] Dívida de pessoas
- [ ] Dívida de automação de testes
- [ ] Dívida de processo
- [ ] Dívida de “build” (construção)
- [ ] Dívida de serviço
- [ ] Dívida de usabilidade
- [ ] Dívida de versionamento
- [ ] Outros (cite quais) { string }

A.2.8 Documentação/representação da dívida técnica

Tendo em vista que documentação ou representação da dívida técnica é a atividade que visa registrar os itens de dívida técnica a fim de um adequado gerenciamento, responda as questões abaixo.

Questão 29: Existe um padrão definido para a documentação ou representação da dívida técnica? *Escala nominal*

- ( ) Sim, o projeto possui um padrão de documentação/representação da dívida técnica
- ( ) Não, a documentação/representação da dívida técnica é feita apenas informalmente

Obs.: A questão 30 só será exibida para o participante caso a resposta da questão 29 tenha sido “sim”.

Questão 30: O padrão de documentação/representação da dívida técnica deve ser aplicado por todos os envolvidos nesta atividade? *Escala nominal*

- ( ) Sim, a aplicação do padrão de documentação/representação da dívida técnica é obrigatória para todos os envolvidos
- ( ) Não, a aplicação do padrão de documentação/representação da dívida técnica é apenas recomendada para os envolvidos

Questão 31: Como os itens de dívida técnica são documentados ou catalogados? *Escala nominal*

- ( ) A dívida técnica é documentada apenas nas notas individuais dos desenvolvedores
- ( ) A dívida técnica é documentada em uma lista de tarefas (backlog) geral, sem detalhes específicos
- ( ) A dívida técnica é documentada em uma lista de tarefas (backlog) específica para itens de dívida
- ( ) Outros (cite quais) { string }

Questão 32: Quais são as ferramentas ou práticas para a documentação da dívida técnica? *Escala nominal*

- [ ] Lista ou backlog de dívida técnica
- [ ] Artefato(s) específico(s) para a documentação da dívida
- [ ] JIRA
- [ ] Wiki
- [ ] Outras práticas/ferramentas (cite quais): { string }

A.2.9 Comunicação da dívida técnica

Sabendo que a comunicação da dívida técnica tem por objetivo tornar visíveis os itens de dívida técnica identificados a todas as partes interessadas, responda as questões abaixo.

Questão 33: Como os itens de dívida técnica não resolvidos são comunicados entre os envolvidos no projeto? *Escala nominal*

- ( ) A dívida técnica é discutida apenas informalmente
- ( ) A dívida técnica é discutida nas reuniões do projeto, mas apenas com algumas partes interessadas envolvidas
- ( ) A dívida técnica é discutida nas reuniões de projeto de forma recorrente, com todas as partes interessadas envolvidas

Questão 34: Quais são as ferramentas ou práticas adotadas para a comunicação da dívida técnica? *Escala nominal*

- [ ] Fóruns de discussão (Wiki do projeto e outras ferramentas de trabalho colaborativo)
- [ ] Inclusão do tópico de dívida técnica em pautas de outras reuniões do projeto
- [ ] Reuniões específicas sobre dívida técnica
- [ ] Outras práticas/ferramentas (cite quais): { string }

A.2.10 Medição da dívida técnica

Medição da dívida técnica é a atividade que visa quantificar a dívida em uma determinada unidade de medida (tempo, valor monetário, etc.) e também avaliar a relação custo/benefício de dívidas técnicas conhecidas em um sistema de software. Baseado nesta definição, responda às seguintes questões.

Questão 35: A medição da dívida técnica identificada é feita de acordo com alguma estratégia definida previamente? *Escala nominal*

- ( ) Sim, existe uma estratégia definida para medição da dívida técnica
- ( ) Não, a medição da dívida técnica é feita informalmente

Questão 36: Como a dívida técnica é medida? *Escala nominal*

- ( ) A medição é feita a partir de informações simples, como número de itens de dívida técnica
- ( ) A medição é feita a partir de uma análise de métricas e indicadores baseando-se em informações específicas sobre o item de dívida técnica
- ( ) A medição é feita de outra forma (se possível, descreva resumidamente como a medição é feita) { string }

Questão 37: Quais informações ou variáveis são utilizadas para a medição dos itens de dívida técnica? *Escala nominal*

- [ ] Pessoas-hora ou Pessoas-mês para a eliminação do item de dívida técnica
- [ ] Custo financeiro para a eliminação do item de dívida técnica
- [ ] Linhas de código do item de dívida técnica
- [ ] Pontos por função do item de dívida técnica
- [ ] Outras informações/variáveis (cite quais) { string }

Questão 38: Quais são as ferramentas ou práticas para apoiar a medição da dívida técnica? *Escala nominal*

- [ ] A medição é feita manualmente, sem auxílio de ferramentas ou práticas
- [ ] SonarQube

- [ ] JIRA
- [ ] Wiki
- [ ] Ferramenta proprietária da organização
- [ ] Outras práticas/ferramentas (cite quais): { string }

A.2.11 Priorização da dívida técnica

Priorização da dívida técnica é a ordenação dos itens de dívida técnica identificados de acordo com regras pré-definidas para apoiar a tomada de decisão de quais itens devem ser pagos primeiramente. Baseado nesta definição, responda as questões abaixo.

Questão 39: Como a dívida técnica é priorizada? *Escala nominal*

- ( ) A dívida é priorizada com base em “palpites” ou estimativas simplificadas com base em experiências anteriores
- ( ) A dívida é priorizada com base em informações de dados históricos de projetos
- ( ) A dívida é priorizada com base em uma tecnologia específica
- ( ) A dívida é priorizada de outra forma (cite) { string }

Questão 40: Quais são as ferramentas ou práticas adotadas para a priorização da dívida técnica? *Escala nominal*

- [ ] Modelos específicos de análise custo/benefício
- [ ] Métodos específicos de apoio a tomada de decisão (“Architecture Tradeoff Analysis Method”, métodos multicritérios, etc.)
- [ ] Classificação de issues
- [ ] Outras práticas/ferramentas (cite quais): { string }

Questão 41: Quais são os principais critérios usados para apoiar a priorização dos itens de dívida técnica? *Escala nominal*

- [ ] Priorização de itens de dívida que mais impactam diretamente no cliente
- [ ] Priorização de itens de dívida que possuem maior nível de gravidade
- [ ] Priorização de itens de dívida que mais impactam o projeto (p.ex., itens de dívida que causam um esforço extra para dar continuidade à evolução do software)
- [ ] Priorização de itens de dívida que estão em partes muito utilizadas do sistema
- [ ] Priorização de itens de dívida que estão em itens do projeto sujeitos a desenvolvimento ou manutenção imediatos

- [ ] Priorização de itens de dívida que exigem menor esforço para serem pagos
- [ ] Priorização de itens de dívida que possuem má relação custo/benefício (é mais caro manter a dívida no projeto do que eliminá-la)
- [ ] Outros critérios (cite) { string }

A.2.12 Pagamento da dívida técnica

Sabendo que a atividade de pagamento da dívida técnica trata da solução ou mitigação da dívida técnica em um sistema de software, responda as questões abaixo.

Questão 42: Existe algum planejamento para o pagamento da dívida técnica? *Escala nominal*

- ( ) Apenas pagamos a dívida técnica quando não é mais possível evitá-la
- ( ) O pagamento da dívida técnica é planejado de acordo com as necessidades do momento ou de acordo com a disponibilidade de tempo
- ( ) O pagamento da dívida técnica é planejado continuamente, com períodos específicos do processo de desenvolvimento destinados a essa atividade

Questão 43: Quais são as técnicas usadas para o pagamento da dívida técnica? *Escala nominal*

- [ ] Refatoração
- [ ] Redesign
- [ ] Reescrita de código
- [ ] Reuniões/Workshops/Treinamentos
- [ ] Outras técnicas (cite quais): { string }

A.2.13 Monitoramento da dívida técnica

Monitoramento da dívida técnica é a atividade de observação das mudanças das relações custo/benefício de dívidas técnicas não solucionadas ao longo do tempo. Baseado nesta definição, responda as questões abaixo.

Questão 44: Como é realizado o monitoramento da dívida técnica? *Escala nominal*

- ( ) O monitoramento é realizado apenas com base no número de itens de dívida técnica, ou é feito ocasionalmente
- ( ) O monitoramento ocorre continuamente, e é realizado com base em vários dados disponíveis sobre os itens de dívida técnica

Questão 45: Quais são as ferramentas ou práticas adotadas para o monitoramento da dívida técnica? *Escala nominal*

- [ ] O monitoramento é feito manualmente
- [ ] SonarQube
- [ ] JIRA
- [ ] Wiki
- [ ] Definição de Pronto / “Definition of Done”
- [ ] Outras práticas/ferramentas (cite quais): { string }

A.2.14 Prevenção da dívida técnica

Sabendo que prevenção da dívida técnica é a atividade que visa prevenir a ocorrência de potenciais itens de dívida técnica, responda às seguintes questões.

Questão 46: Existem práticas padronizadas pela organização ou pelo gerente do projeto para prevenção da dívida técnica? *Escala nominal*

- ( ) Sim, existem práticas definidas para a prevenção da dívida
- ( ) Não, a prevenção é feita apenas informalmente por cada envolvido

Obs.: A questão 47 só será exibida para o participante caso a resposta da questão 46 tenha sido “sim”.

Questão 47: As práticas padronizadas para prevenção da dívida técnica existentes precisam ser aplicadas por todos os envolvidos? *Escala nominal*

- ( ) Sim, as práticas padrão para prevenção da dívida técnica são obrigatórias para todos os envolvidos
- ( ) Não, as práticas padrão para prevenção da dívida técnica são apenas recomendadas

Questão 48: Quais são as ferramentas ou práticas adotadas para a prevenção da dívida técnica? *Escala nominal*

- [ ] “Guidelines” (conjunto de orientações de boas práticas estabelecidas)
- [ ] Padrões de codificação
- [ ] Revisões de código
- [ ] Reuniões de retrospectiva
- [ ] Definição de Pronto / “Definition of Done”
- [ ] Outras práticas/ferramentas (cite quais): { string }

A.2.15 Outra atividade de gerenciamento da dívida técnica

Baseado na atividade de gerenciamento da dívida técnica adicional que você citou previamente, responda às seguintes perguntas.

Questão 49: Existe uma prática formalizada para a execução desta atividade? *Escala nominal*

- ( ) Sim, o projeto possui uma prática formalizada para o gerenciamento desta atividade
- ( ) Não, a atividade é executada apenas informalmente

Questão 50: Esta atividade tem sua execução obrigatória para todos os envolvidos? *Escala nominal*

- ( ) Sim, a aplicação da estratégia desta atividade de gerenciamento é obrigatória para todos os envolvidos
- ( ) Não, a aplicação da estratégia desta atividade de gerenciamento é apenas recomendada para todos os envolvidos

Questão 51: Em que momento do projeto de software esta atividade é executada? *Escala nominal*

- ( ) Não ocorre em um momento definido, sendo executada sempre que algum problema é encontrado
- ( ) A atividade é executada sempre ao final de cada iteração/sprint
- ( ) A atividade é executada continuamente, ou seja, ocorre ao longo de todo o processo de desenvolvimento

Questão 52: Cite abaixo quais são as ferramentas ou técnicas que são usadas para a execução desta atividade. *Escala nominal*

- { string }

Questão 53: Use a caixa de texto abaixo para acrescentar quaisquer informações que julgar necessárias sobre a atividade de gerenciamento da dívida técnica. *Escala nominal*

- { string }

A.2.16 Encerramento

Obrigado pela participação! Caso tenha dúvidas ou sugestões, ou deseje obter mais informações sobre o assunto após a divulgação dos resultados, entre em contato com Victor Machado da Silva, através do e-mail victorms@cos.ufrj.br.

A.3 English version

A.3.1 Introduction

This research aims to observe how the technical debt management is being treated in the context of Brazilian software organizations. Its primary purpose is to characterize the industry's perspective regarding the identification, measurement, prioritization and other technical debt management activities in software projects and compare with the recent research on the subject.

The results of this research have high relevance for the Brazilian software industry, as it attempts to identify technical debt management strategies and technologies that can be improved or applied in the software development context in Brazil.

The participation of the software professional in this survey is voluntary and anonymous. However, your participation is of most relevance and importance! More significant participation from the industry's practitioners turns into more significant results and contributions of this study to the software community.

Two sets of questions compose the questionnaire: the participant's characterization, his/her organization and the project he/she is inserted, which takes about five minutes; and a technical questionnaire, involving questions related to technical debt and its management. For the second part, the estimated response time depends on the participant's previous experiences and may take from five to twenty minutes.

This survey is inserted on the research and development activities related to the Masters' dissertation conducted by Victor Machado da Silva, at Universidade Federal do Rio de Janeiro – COPPE/UFRJ (Rio de Janeiro, Brazil), under the supervision of prof. Guilherme Horta Travassos.

This research adopts the ethical and scientific principles that support the scientific research on empirical software engineering. As of that, personal and sensitive data are not requested, and the participation is entirely anonymous and voluntary. All results are going to be presented in an aggregated form, without the possibility of the participant's identification.

If you have doubts or suggestions, please contact through the e-mail victorms@cos.ufrj.br.

A.3.2 Participant's characterization

This section aims to obtain some professional information about you.

Question 1: What is your role in the current or most recent software project you have participated? Please choose the options that apply. *Nominal scale*

- [ ] Project Manager

- [ ] Product Line Manager
- [ ] Team Leader
- [ ] Software Architect
- [ ] Software Test Engineer
- [ ] Programmer
- [ ] User Interface Designer
- [ ] Requirements Analyst
- [ ] Researcher
- [ ] Other { string }

Question 2: What is your academic formation? *Nominal scale*

- ( ) Complete technical/high school
- ( ) Incomplete undergraduate
- ( ) Undergraduate degree
- ( ) Specialist degree
- ( ) Master's degree
- ( ) Doctor's degree

Question 3: What is your experience, in work years, with software projects? Only numbers can be used in this field. *Ratio scale*

- { positive integers }

A.3.3 Organization characterization

The purpose of this section is to gather some information about your organization. Please answer about the current or most recent organization you have worked.

Question 4: What is your organization's field? *Nominal scale*

- ( ) Information Technology
- ( ) Telecommunications
- ( ) Medical/Biological/Pharmaceutical
- ( ) Financing (Insurances, Banking, etc.)
- ( ) Government
- ( ) Consulting
- ( ) Education
- ( ) Others { string }

Question 5: What is the size of your organization, in a number of employees? *Interval scale*

- ( ) Fewer than ten employees
- ( ) Between 10 and 49 employees

- ( ) Between 50 and 99 employees
- ( ) More than 100 employees
- ( ) I do not know

Question 6: Does your organization have any current maturity evaluation of software processes (MPS.BR, CMMI, etc.)? *Nominal scale*

- [ ] It has MPS.BR evaluation
- [ ] It has CMMI evaluation
- [ ] It has other evaluations
- [ ] I do not know
- [ ] It does not have any evaluation

Obs.: In case the participant answers “I do not know” or “it does not have any evaluation,” the three first choices must be disabled. Questions 7, 8 and 9 will be exhibited if the participant only answers “it has MPS.BR evaluation”, “it has CMMI evaluation” or “it has other evaluations,” respectively.

Question 7: What is your organization’s MPS.BR level? Answer this question only if you have answered “it has MPS.BR evaluation” on question 6. *Ordinal scale*

- ( ) A
- ( ) B
- ( ) C
- ( ) D
- ( ) E
- ( ) F
- ( ) G

Question 8: What is your organization’s CMMI level? Answer this question only if you have answered “it has CMMI evaluation” on question 6. *Ordinal scale*

- ( ) 1
- ( ) 2
- ( ) 3
- ( ) 4
- ( ) 5

Question 9: Name the evaluations on maturity models (and their levels, if any) by your organization *Nominal scale*

- { string }

A.3.4 Project characterization

This section aims to obtain information about the project that will be used as a reference to the remaining questions of this survey. When the question mentions “your most recent project,” please answer considering the most recent project you have worked, or that is currently working.

Question 10: What is the problem domain problem of your most recent project? *Nominal scale*

- ( ) Telecommunications
- ( ) Education
- ( ) Government
- ( ) Image
- ( ) Retail, distribution, and transport
- ( ) Financing
- ( ) Medical systems
- ( ) Defense
- ( ) Pharmaceuticals
- ( ) Outros { string }

Question 11: What is your most recent software project’s lifecycle model? *Nominal scale*

- ( ) Waterfall
- ( ) Incremental
- ( ) Spiral
- ( ) Agile
- ( ) Others { string }

A.3.5 Understanding and perception of technical debt

This part of the survey aims to gather the general understanding regarding the technical debt definition and its aspects.

Question 12: Do you know what the term “technical debt” means? *Nominal scale*

- ( ) Yes
- ( ) No
- ( ) Yes, but I know it by another name (a name which) { string }

Obs.: The remaining questionnaire will only be exhibited in case the answer to question 12 is “yes” or “yes, but I know it by another name.”

Question 13: In your opinion, which of the below items can be associated with the technical debt concept? Select all that you consider relevant. *Nominal scale*

- [ ] Defects
- [ ] Required but unimplemented features
- [ ] Lack of support processes in the execution of the project
- [ ] Planned, but not realized or not finished, tasks (e.g., requirement specification, models, test plans, test execution, etc.)
- [ ] Trivial code quality issues, that do not violate code rules
- [ ] Issues associated with low external quality, like usability and efficiency
- [ ] Issues associated with low internal quality, like maintainability and reusability
- [ ] Architectural issues (like modularity violation)
- [ ] "Shortcuts" taken during design
- [ ] Poorly written code, that violates the code rules
- [ ] Presence of known defects, which were not eliminated
- [ ] Code smells

Question 14: In your most recent project, did you notice any problem that could be associated with the technical debt concept? *Nominal scale*

- ( ) Yes
- ( ) No

Obs.: The remaining questionnaire will only be exhibited in case the answer to question 14 is "yes."

Question 15: Does your organization adopt technical debt management activities, even if partially? *Nominal scale*

- ( ) Yes
- ( ) No

Question 16: Do you or your manager adopt, in your most recent project, any technical debt management activities, besides those already adopted by your organization? *Nominal scale*

- ( ) Yes
- ( ) No

Obs.: The remaining questionnaire will only be exhibited in case the answer to questions 15 or 16 is "yes."

A.3.6 Technical debt management

The purpose of this group of questions is to identify, in a general matter, the adoption and relevance of technical debt management practices in the Brazilian software industry's projects. By "technical debt management," consider all activities that organize, monitor and control the technical debt and its impacts on software projects.

To all the questions, please answer based on your most recent project.

Question 17: select which activities were adopted by your most recent project, either through a formal strategy of through informal practices. *Nominal scale*

- [ ] Technical debt identification – aims to detect technical debt items on software artifacts, caused by intentional or unintentional technical decisions
- [ ] Technical debt documentation/representation – aims to register the technical debt items, to enable proper management
- [ ] Technical debt communication – attempts to turn visible the identified technical debt items to all stakeholders
- [ ] Technical debt measurement – addresses the quantification of identified technical debt in some measurement unit (time, monetary value, etc.), and evaluates the cost/benefit ratio of known technical debt items in a software system
- [ ] Technical debt prioritization – aims to order the identified technical debt items according to predefined rules, to support the decision-making process of which items should be repaid first
- [ ] Technical debt repayment – deals with the solution or mitigation of the technical debt item in a software system
- [ ] Technical debt monitoring – attempts to observe changes in the cost/benefit ratio of known technical debt items throughout time
- [ ] Technical debt prevention – aims to prevent the occurrence of potential technical debt items
- [ ] Other activities

Obs.: Question 18 will only be exhibited if the answer "other activities" is selected.

Question 18: Name one of the technical debt management activities adopted by your most recent project, that was not mentioned in the previous question. *Nominal scale*

- { string }

Obs.: Questions 19 and 20 will only have as options the technical debt management activities selected by the participant on question 17.

Question 19: For each activity listed below, select the respective responsibilities for your most recent project. *Nominal scale*

	Project manager	Team leader	Software architect	Dev. team	None of the above
Technical debt identification	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt documentation/representation	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt communication	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt measurement	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt prioritization	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt repayment	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt monitoring	[ ]	[ ]	[ ]	[ ]	[ ]
Technical debt prevention	[ ]	[ ]	[ ]	[ ]	[ ]
Other activities previously mentioned	[ ]	[ ]	[ ]	[ ]	[ ]

Question 20: In case there are other responsible roles for the listed activities, besides the ones indicated in the previous answer, please fill the fields below. If there is no other responsible, write “Nothing to declare.” *Nominal scale*

- Technical debt identification { string }
- Technical debt documentation/representation { string }
- Technical debt communication { string }
- Technical debt measurement { string }
- Technical debt prioritization { string }
- Technical debt repayment { string }
- Technical debt monitoring { string }
- Technical debt prevention { string }
- Other activities previously mentioned { string }

Question 21: Order the activities below, according to your opinion on the most important to your most recent project. Consider that “1” is the least important activity, and “8” is the most important activity. It is not possible to assign two activities to the same order, i.e., all activities must have a distinct importance order. *Nominal scale*

- Technical debt identification { number between 1 and 8 }
- Technical debt documentation/representation { number between 1 and 8 }
- Technical debt communication { number between 1 and 8 }
- Technical debt measurement { number between 1 and 8 }
- Technical debt prioritization { number between 1 and 8 }
- Technical debt repayment { number between 1 and 8 }
- Technical debt monitoring { number between 1 and 8 }
- Technical debt prevention { number between 1 and 8 }

Obs.: The following sections will only be exhibited if the participant has selected the appropriate option in question 17. For example, if s/he answered that his/her project adopts technical debt identification and monitoring activities than those are the only two sections available to answer.

A.3.7 Technical debt identification

This activity aims to detect technical debt items caused by intentional or unintentional technical decisions. Based on this definition, answer the following questions, using your most recent project as a reference.

Question 22; Is there a formal approach to the technical debt identification? *Nominal scale*

- ( ) Yes, the project has a formal approach to the technical debt identification
- ( ) No, the technical debt identification is conducted informally

Obs.: Question 23 will only be exhibited if the answer to question 22 is “yes.”

Question 23: Does the technical debt identification approach have to be applied by all the people involved in this activity? *Nominal scale*

- ( ) Yes, the application of the approach is mandatory to all involved
- ( ) No, the application of the approach is only optional

Question 24: When if the technical debt identified? *Nominal scale*

- ( ) There’s no specific moment; we identify the technical debt only occasionally throughout the project

- ( ) There's no specific moment; we identify the technical debt whenever a problem comes up
- ( ) We always identify the technical debt at the end of each iteration/sprint
- ( ) The technical debt identification is a continuous activity, occurring throughout all development process

Question 25: Which tools or techniques are used to identify the technical debt? *Nominal scale*

- [ ] Manual code inspection
- [ ] Dependency analysis
- [ ] Checklist
- [ ] SonarQube/SQALE
- [ ] CheckStyle
- [ ] FindBugs
- [ ] CodeVizard
- [ ] CLIO
- [ ] Other tools/techniques (name which) { string }

Question 26: Is the identified technical debt classified? *Nominal scale*

- ( ) Yes, we have a set of classification criteria to organize the identified technical debt items better
- ( ) No, we only identify the technical debt, and there's no classification

Obs.: Question 27 will only be exhibited if the answer to question 26 is "yes."

Question 27: How the technical debt is classified? Select all options that apply. *Nominal scale*

- [ ] Technical debt type (architecture debt, code debt, etc.)
- [ ] Technical debt dimension (intentional/unintentional, strategical/tactical, etc.)
- [ ] Origin artifact (code, diagrams, requirements, etc.)
- [ ] We have our classification criteria (if possible, give a brief description) { string }

Obs.: Question 28 will only be exhibited if the answer to question 27 is "technical debt type."

Question 28: If the technical debt is classified according to its type, select below which ones are usually identified. *Nominal scale*

- [ ] Design debt
- [ ] Architecture debt

- [ ] Documentation debt
- [ ] Test debt
- [ ] code debt
- [ ] Defect debt
- [ ] Requirement debt
- [ ] Infrastructure debt
- [ ] People debt
- [ ] Test automation debt
- [ ] Process debt
- [ ] Build debt
- [ ] Service debt
- [ ] Usability debt
- [ ] Versioning debt
- [ ] Others (name which) { string }

A.3.8 Technical debt documentation/representation

Answer the following questions, considering that technical debt documentation/representation is the activity that aims to register the technical debt items, intending to proper management.

Question 29: Is there a standard for the technical debt documentation? *Nominal scale*

- ( ) Yes, we have a standard for technical debt documentation in our project
- ( ) No, the documentation is informal

Obs.: Question 30 will only be exhibited if the answer to question 29 is “yes.”

Question 30: Is the standard for the technical debt documentation/representation mandatory? *Nominal scale*

- ( ) Yes, it is mandatory for all
- ( ) No, it is only recommended to the people involved in this activity

Question 31: How the technical debt items are documented or cataloged? *Nominal scale*

- ( ) It is documented only in the individual notes of the developers
- ( ) It is documented in a general backlog, without specific details
- ( ) It is documented in a specific technical debt backlog
- ( ) Others (please mention them) { string }

Question 32: What are the tools or practices of the technical debt documentation? *Nominal scale*

- [ ] Technical debt list or backlog
- [ ] Specific artifact(s) to debt documentation
- [ ] JIRA
- [ ] Wiki
- [ ] Other tools/practices (please mention them) { string }

A.3.9 Technical debt communication

Technical debt communication aims to make visible the known technical debt items to all stakeholders. With this definition, please answer the following questions.

Question 33: How the unresolved technical debt items are communicated to the project stakeholders? *Nominal scale*

- ( ) It is only discussed informally
- ( ) It is discussed in project meetings, but only with some stakeholders
- ( ) It is discussed in project meetings, with all stakeholders

Question 34: What are the tools or practices adopted to the technical debt communication? *Nominal scale*

- [ ] Discussion forums (project Wiki and other tools of collaborative work)
- [ ] Inclusion of the technical debt topic in project meetings
- [ ] Specific technical debt meetings
- [ ] Other practices/tools (please mention them) { string }

A.3.10 Technical debt measurement

Technical debt measurement is the activity that aims to quantify the debt in a specific measurement unit (time, monetary value, etc.). Also, it aims to evaluate the cost/benefit relation of known technical debt items in a software system. Based on this definition, please answer the questions below.

Question 35: Is the known technical debt measured according to any previously defined strategy? *Nominal scale*

- ( ) Yes, we have a strategy for the technical debt measurement
- ( ) No, the technical debt measurement is an informal activity

Question 36: How is the technical debt measured? *Nominal scale*

- ( ) The measurement is made based on simple information, like the number of technical debt items
- ( ) The measurement is made based on a series of metrics and indicators obtained through specific information about the technical debt item

- ( ) We have other forms to measure the technical debt (if possible, please give a brief description) { string }

Question 37: What information or variable are used to measure the technical debt items? *Nominal scale*

- [ ] Person-hours or person-months to eliminate the technical debt item
- [ ] Financial cost to eliminate the technical debt item
- [ ] LOC of the technical debt item
- [ ] Function points of the technical debt item
- [ ] Other information/variables (name them) { string }

Question 38: Which tools or practices are used to support the technical debt measurement? *Nominal scale*

- [ ] The measurement is manual, without tool support
- [ ] SonarQube
- [ ] JIRA
- [ ] Wiki
- [ ] Proprietary tool
- [ ] Other practices/tools (please name them) { string }

A.3.11 Technical debt prioritization

Technical debt prioritization is the ordination of the technical debt items, according to pre-established rules, to support the decision-making on which items should be repaid first. Based on this definition, please answer the following questions.

Question 39: How is the technical debt prioritized? *Nominal scale*

- ( ) Based on “hunches” or in simplified estimates using previous experiences
- ( ) Based on historical data of the project
- ( ) Based on a specific technology
- ( ) Based on other forms (please describe, if possible) { string }

Question 40: Which tools or practices are used to the technical debt prioritization? *Nominal scale*

- [ ] Specific cost/benefit analysis models
- [ ] Specific decision-making methods (Architecture tradeoff analysis method, multicriteria methods, etc.)
- [ ] Issue classification
- [ ] Other practices/tools (please name them) { string }

Question 41: What are the main criteria adopted to support the technical debt prioritization? *Nominal scale*

- [ ] Prioritization of the items that most impact the client
- [ ] Prioritization of the items that have the highest level of severity
- [ ] Prioritization of the items that most impact the project (e.g., items that cause an extra effort to keep the software evolution)
- [ ] Prioritization of the items that are in used parts of the system
- [ ] Prioritization of the items that are in parts of the project subject to immediate development or maintenance
- [ ] Prioritization of the items that demand less effort to be paid
- [ ] Prioritization of the items that have poor cost/benefit relation (it is more expensive to keep the debt than to eliminate it)
- [ ] Others (please describe) { string }

A.3.12 Technical debt repayment

Technical debt repayment is the solution or mitigation of the technical debt item. Based on this definition, please answer the following questions.

Question 42: Is there any planning for the technical debt repayment? *Nominal scale*

- ( ) We only repay it when it is not possible to avoid it anymore
- ( ) We plan the repayment according to the moment's necessity or according to the time availability
- ( ) We have a continuous plan to repay it, with specific periods of the development process destined for this activity

Question 43: What techniques are used for the technical debt repayment? *Nominal scale*

- [ ] Refactoring
- [ ] Redesign
- [ ] Code rewriting
- [ ] Meetings/Workshops/Trainings
- [ ] Other techniques (please name them) { string }

A.3.13 Technical debt monitoring

Technical debt monitoring aims to observe the changes in the cost/benefit relations of the unsolved technical debt items throughout time. Based on this definition, answer the following questions.

Question 44: How it the technical debt monitored? *Nominal scale*

- ( ) The monitoring is conducted only based on the number of technical debt items, or it is occasionally conducted
- ( ) The debt is continuously monitored and is conducted based on several available data about the technical debt items

Question 45: What are the tools or practices adopted to the technical debt monitoring? *Nominal scale*

- [ ] The monitoring is manual
- [ ] SonarQube
- [ ] JIRA
- [ ] Wiki
- [ ] Definition of Done
- [ ] Other practices/tools (name them) { string }

A.3.14 Technical debt prevention

Technical debt prevention is the activity that aims to prevent the occurrence of potential technical debt items. Based on this definition, please answer the following questions.

Question 46: Are there formal practices to conduct the technical debt prevention? *Nominal scale*

- ( ) Yes, we have formal practices to prevent the technical debt
- ( ) No, the technical debt prevention is conducted informally by each project member

Obs.: Question 47 will only be exhibited if the answer to question 46 is “yes.”

Question 47: Do the formal practices to prevent the technical debt must be applied to all stakeholders? *Nominal scale*

- ( ) Yes, the practices are mandatory to all involved
- ( ) No, the practices are only recommended

Question 48: What are the tools or practices adopted to the technical debt prevention? *Nominal scale*

- [ ] Guidelines
- [ ] Code patterns
- [ ] Code reviews
- [ ] Retrospective meetings
- [ ] Definition of Done
- [ ] Other practices/tools (name them) { string }

A.3.15 Other technical debt management activity

Based on the additional technical debt management activity that you previously mentioned, please answer the following questions.

Question 49: Is there a formal approach to conduct this activity? *Nominal scale*

- ( ) Yes, we have a formal approach
- ( ) No, the activity is conducted informally

Question 50: Is this activity mandatory for all the involved project members? *Nominal scale*

- ( ) Yes, the activity is mandatory for all participants
- ( ) No, the activity is optional

Question 51: When is this activity conducted? *Nominal project*

- ( ) There's no defining moment; we conduct whenever we find a problem
- ( ) We conduct it at the end of every iteration/sprint
- ( ) The activity is conducted continuously, throughout all development process

Question 52: Name which tools or techniques are used to conduct this activity. *Nominal scale*

- { string }

Question 53: Use the text box below to add any information that you would like to include the technical debt management activity. *Nominal scale*

- { string }

A.3.16 Closing

Thank you for your participation! In case you have any doubts or suggestions, or if you want to obtain more information about the subject after the dissemination of results, please contact Victor Machado da Silva, through victorms@cos.ufrj.br.

Appendix B – Survey Results

This appendix presents the results obtained from the participants, extracted from the Limesurvey platform.

B.1 Introduction

In this appendix, we present the results obtained from the participants. The results were extracted through the “export” tool from *Limesurvey*. The answers were translated into English, and the presentation was optimized to provide better visualization.

Some information obtained from *Limesurvey*, such as “Sent date,” “Last page,” etc., are not reproduced here since it is not relevant to the study. Whenever “N/A indicates a field” or “-,“ the participant did not provide the information. It means that s/he did not finish the survey, and then did not answer that section; or that the platform did not exhibit that section to him/her.

Questions with multiple answers are represented with the participant’s choices separated by semicolons. Questions in which no participant have answered are not presented here.

B.2 Incomplete answers

This section presents the results gathered by the participants that did not complete the questionnaire. No participant from this group answered questions 23 onwards. Thus they are not represented here.

From the 27 participants that did not complete the survey, i.e., they did not answer all the required questions, 18 did not answer any questions, and are not represented here.

Participant ID	Participant's characterization			Organization characterization			
	Question 1	Question 2	Question 3	Question 4	Question 5	Question 6	Question 9
33	Team Leader; Programmer	Undergraduate degree	9	Information Technology	Between 10 and 49 employees	It does not have any evaluation	-
36	Programmer	Incomplete undergraduate	1	Consulting	Between 10 and 49 employees	I do not know	-
37	Programmer	Undergraduate degree	15	Education	More than 100 employees	It does not have any evaluation	-
41	Project Manager; Team Leader	Undergraduate degree	6	Information Technology	More than 100 employees	It does not have any evaluation	-
45	Programmer	Undergraduate degree	0	Information Technology	Fewer than ten employees	I do not know	-
61	Programmer	Incomplete undergraduate	2	Information Technology	Between 10 and 49 employees	It has other evaluations	PMO
65	Others: Sales Manager	Specialist degree	22	Telecommunications	More than 100 employees	I do not know	-
86	Project Manager	Specialist degree	1	Information Technology	Between 10 and 49 employees	It does not have any evaluation	-

Participant ID	Project characterization		Understanding and perception of technical debt				
	Question 10	Question 11	Question 12	Question 13	Question 14	Question 15	Question 16
33	Others: Project Management	Agile	Yes	Trivial code quality issues that do not violate code rules; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Code smells	-	-	-
36	-	-	-	N/A	-	-	-
37	Education	Agile	Yes	N/A	-	-	-
41	Retail, distribution, and transport	Agile	Yes	Defects; Issues associated with low external quality, like usability and efficiency; “Shortcuts” taken during design; Presence of known defects, which were not eliminated	Yes	Yes	No
45	-	-	-	N/A	-	-	-
61	Others: Serious games	Agile	Yes	N/A	-	-	-
65	-	-	-	N/A	-	-	-
86	Retail, distribution, and transport	Agile	Yes	Trivial code quality issues, that do not violate code rules; Issues associated with low external quality, like usability and efficiency; Issues associated with low internal quality, like maintainability and reusability; Architectural issues (like modularity violation); “Shortcuts” taken during design; Presence of known defects, which were not eliminated; Code smells	Yes	No	Yes

Participant ID	Technical debt management					
	Question 17	Question 19				Question 20
		TD Identification	TD Documentation/Representation	TD Prioritization	TD Repayment	
33	N/A	-	-	-	-	-
36	N/A	-	-	-	-	-
37	N/A	-	-	-	-	-
41	N/A	-	-	-	-	-
45	N/A	-	-	-	-	-
61	N/A	-	-	-	-	-
65	N/A	-	-	-	-	-
86	TD Identification; TD Documentation/Representation; TD Prioritization; TD repayment	Dev. team	Project manager; Software architect	Software architect	Software architect; Dev. team	Nothing to declare

Participant ID	Technical debt management								Technical debt identification
	Question 21								Question 22
	TD Identification	TD Documentation/ Representation	TD Communication	TD Measurement	TD Prioritization	TD Repayment	TD Monitoring	TD Prevention	
33	-	-	-	-	-	-	-	-	-
36	-	-	-	-	-	-	-	-	-
37	-	-	-	-	-	-	-	-	-
41	-	-	-	-	-	-	-	-	-
45	-	-	-	-	-	-	-	-	-
61	-	-	-	-	-	-	-	-	-
65	-	-	-	-	-	-	-	-	-
86	8	5	8	5	7	8	4	4	No

B.2 Complete answers

This section presents the results gathered by the 35 participants that completed the questionnaire. Questions 49 onwards and others that were not answered by any participant from this group are not represented here.

To further simplify the representation, questions 19 onwards are only documented here for the participants that answered them. If the participant answered “no” to question 12 (if s/he knows what “technical debt” means), then all of his/her answers starting from questions 13 are going to be either “N/A” or “-.“ Additionally, if s/he answers that no TDM activities were conducted in his/her project, then all answers from questions 17 onwards are going to be either “N/A” or “-.“

Some answers were simplified to achieve a better representation. Their meanings, however, are preserved.

Participant ID	Participant's characterization			Organization characterization	
	Question 1	Question 2	Question 3	Question 4	Question 5
19	Project Manager; Researcher	Master's degree	13	Government	More than 100 employees
20	Software Test Engineer; Require-ments Analyst	Specialist degree	9	Others: Oil & Gas/Distributing	More than 100 employees
21	Team Leader	Specialist degree	15	Information Technology	More than 100 employees
22	Project Manager; Requirements Analyst	Master's degree	3	Government	More than 100 employees
23	Programmer; Requirements Analyst	Specialist degree	11	Government	More than 100 employees
24	Others: Quality Analyst	Master's degree	43	Government	More than 100 employees
25	Others: Scrum Master	Specialist degree	11	Government	More than 100 employees
26	Project Manager; Team Leader; Requirements Analyst	Specialist degree	10	Information Technology	Between 10 and 49 employ-ees
28	Software Test Engineer	Undergraduate degree	1	Consulting	More than 100 employees
29	Product Line Manager; Software Architect	Master's degree	30	Telecommunications	Between 10 and 49 employ-ees
30	Programmer	Incomplete undergraduate	1	Others: Software Engineering	More than 100 employees
38	Project Manager; Team Leader; Software Architect; Programmer	Undergraduate degree	20	Information Technology	Between 10 and 49 employ-ees
39	Team Leader; Programmer	Specialist degree	12	Information Technology	Between 10 and 49 employ-ees
42	Software Test Engineer; Program-mer	Undergraduate degree	10	Information Technology	Fewer than ten employees
44	Researcher	Doctorate	20	Financing (Insurances, Banking, etc.)	More than 100 employees
46	Project Manager	Master's degree	10	Consulting	More than 100 employees
52	Project Manager	Doctorate	25	Information Technology	More than 100 employees
54	Requirements Analyst; Researcher	Doctorate	28	Financing (Insurances, Banking, etc.)	More than 100 employees
55	Project Manager	Master's degree	10	Telecommunications	Fewer than ten employees
56	Project Manager	Doctorate	20	Government	More than 100 employees
57	Project Manager; Researcher	Master's degree	19	Information Technology	Between 10 and 49 employ-

Participant ID	Participant's characterization			Organization characterization	
	Question 1	Question 2	Question 3	Question 4	Question 5
					ees
58	Project Manager	Master's degree	11	Information Technology	Between 10 and 49 employ-ees
59	Programmer	Undergraduate degree	2	Information Technology	Between 10 and 49 employ-ees
60	Team Leader; Researcher	Master's degree	5	Information Technology	Between 10 and 49 employ-ees
63	Researcher	Doctorate	15	Information Technology	More than 100 employees
66	Software Test Engineer	Undergraduate degree	35	Government	More than 100 employees
71	Software Test Engineer	Master's degree	13	Information Technology	More than 100 employees
72	Software Architect	Specialist degree	12	Information Technology	More than 100 employees
74	Software Architect	Undergraduate degree	13	Information Technology	More than 100 employees
77	Product Line Manager	Undergraduate degree	40	Information Technology	More than 100 employees
83	Others: Technology Coordinator	Specialist degree	12	Others: Media	More than 100 employees
84	Others: Scrum Master	Incomplete undergraduate	14	Information Technology	More than 100 employees
85	Others: Business Analyst	Incomplete undergraduate	19	Information Technology	More than 100 employees
87	Project Manager	Master's degree	10	Information Technology	More than 100 employees
90	Project Manager	Specialist degree	11	Information Technology	Fewer than ten employees

Participant ID	Organization characterization				Project characterization		Understanding and perception of technical debt
	Question 6	Question 7	Question 8	Question 9	Question 10	Question 11	Question 12
19	It does not have any evaluation	-	-	-	Government	Agile	Yes
20	It does not have any evaluation	-	-	-	Others: Fuel Distribution	Spiral	Yes
21	It does not have any evaluation	-	-	-	Government	Agile	Yes
22	It has MPS.BR evaluation	G	-	-	Defense	Incremental	Yes
23	It does not have any evaluation	-	-	-	Government	Waterfall	Yes
24	It does not have any evaluation	-	-	-	Government	Agile	Yes
25	It does not have any evaluation	-	-	-	Government	Agile	Yes
26	It has other evaluations	-	-	QPS, ISO, TECH seal	Retail, distribution, and transport	Agile	Yes
28	It has CMMI evaluation	-	5	-	Retail, distribution and transport	Incremental	No
29	It does not have any evaluation	-	-	-	Telecommunications	Agile	Yes
30	It has CMMI evaluation	-	5	-	Others: Business	Agile	No
38	It does not have any evaluation	-	-	-	Retail, distribution, and transport	Agile	No
39	I do not know	-	-	-	Others: Construction	Agile	No
42	I do not know	-	-	-	Others: Construction	Waterfall	No
44	It does not have any evaluation	-	-	-	Others: Credit	Incremental	Yes
46	I don’t know	-	-	-	Government	Agile	No
52	It does not have any evaluation	-	-	-	Government	Agile	Yes

Participant ID	Organization characterization				Project characterization		Understanding and perception of technical debt
	Question 6	Question 7	Question 8	Question 9	Question 10	Question 11	Question 12
54	It does not have any evaluation	-	-	-	Financing	Agile	Yes
55	It does not have any evaluation	-	-	-	Telecommunications	Agile	No
56	It doesn't have any evaluation	-	-	-	Telecommunications	Incremental	No
57	It does not have any evaluation	-	-	-	Others: Industry	Agile	No
58	I do not know	-	-	-	Others: Industry 4.0	Agile	No
59	I do not know	-	-	-	Others: Business	Agile	No
60	It does not have any evaluation	-	-	-	Image	Agile	No
63	It has other evaluations	-	-	Test	Government	Spiral	No
66	It has other evaluations	-	-	MPT-BR level 1	Financing	Incremental	No
71	It has CMMI evaluation	-	2	-	Government	Incremental	Yes
72	It has CMMI evaluation	-	2	-	Government	Agile	Yes
74	I do not know	-	-	-	Government	Agile	Yes
77	It does not have any evaluation	-	-	-	Government	Agile	No
83	I don't know	-	-	-	Image	Agile	Yes
84	It does not have any evaluation	-	-	-	Medical systems	Agile	Yes
85	I do not know	-	-	-	Retail, distribution, and transport	Waterfall	Yes
87	It does not have any evaluation	-	-	-	Defense	Agile	No
90	It does not have any evaluation	-	-	-	Government	Incremental	Yes

Participant ID	Understanding and perception of technical debt			
	Question 13	Question 14	Question 15	Question 16
19	Defects; Required but unimplemented features; Lack of support processes to the execution of the project; Planned, but not realized or not finished, tasks; Trivial code quality issues, that do not violate code rules; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects; Code smells	Yes	No	No
20	Planned, but not realized or not finished, tasks; Issues associated with low internal quality; Poorly written code; Presence of known defects	Yes	Yes	No
21	Defects; Planned, but not realized or not finished, tasks; Trivial code quality issues, that do not violate code rules; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects; Code smells	Yes	No	No
22	Issues associated with low internal quality	No	-	-
23	Required but unimplemented features; Lack of support processes to the execution of the project; Planned, but not realized or not finished, tasks; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects; Code smells	Yes	No	No
24	Defects; Planned, but not realized or not finished, tasks; Trivial code quality issues, that do not violate code rules; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; Poorly written code; Presence of known defects	Yes	Yes	Yes
25	Defects; Trivial code quality issues, that do not violate code rules; Poorly written code	No	-	-
26	Defects; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects	Yes	Yes	Yes
29	Issues associated with low internal quality; Architectural issues; Poorly written code; Presence of known defects	Yes	Yes	No
44	Defects; Lack of support processes to the execution of the project; Planned, but not realized or not finished, tasks; Trivial code quality issues, that do not violate code rules; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects; Code smells	Yes	No	No
52	Lack of support processes to the execution of the project; Planned, but not realized or not finished, tasks; Issues associated with low external quality; Issues associated with low	Yes	No	Yes

Participant ID	Understanding and perception of technical debt			
	Question 13	Question 14	Question 15	Question 16
	internal quality; “Shortcuts” taken during design; Poorly written code; Presence of known defects			
54	Required but unimplemented features; Lack of support processes in the execution of the project	No	-	-
71	Planned, but not realized or not finished, tasks; Issues associated with low internal quality; “Shortcuts” taken during design; Presence of known defects	Yes	No	No
72	Planned, but not realized or not finished, tasks; Issues associated with low external quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects; Code smells	Yes	No	No
74	Planned, but not realized or not finished, tasks; Trivial code quality issues, that do not violate code rules; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Presence of known defects	Yes	No	Yes
83	Issues associated with low external quality; Issues associated with low internal quality; “Shortcuts” taken during design; Poorly written code; Code smells	Yes	Yes	Yes
84	Trivial code quality issues that do not violate code rules; Issues associated with low external quality; Issues associated with low internal quality; Architectural issues; “Shortcuts” taken during design; Poorly written code; Code smells	Yes	Yes	Yes
85	Issues associated with low internal quality; “Shortcuts” taken during design; Presence of known defects	No	-	-
90	Defects; Required but unimplemented features; Planned, but not realized or not finished, tasks; Trivial code quality issues; Code smells	Yes	No	No

Participant ID	Technical debt management	Technical debt management						
	Question 17	Question 19						
		TD Identification	TD Documentation/ Representation	TD Communication	TD Measurement	TD Prioritization	TD Repayment	TD Prevention
20	TD Communication; TD Prioritization; TD Repayment	-	-	Team leader; Dev. team	-	Project manager; Software architect	Team leader; Dev. team	-
24	TD Prevention	-	-	-	-	-	-	Team leader
26	TD Identification; TD Documentation/Representation; TD Repayment	None of the above	Project manager	-	-	-	Dev. team	-
29	TD Documentation/Representation; TD Communication; TD Prioritization; TD Prevention	-	Project manager; Team leader	Team leader; Dev. team	-	Project manager; team leader	-	Project manager; Team leader; Software architect; Dev. team
52	TD Identification	Team leader	-	-	-	-	-	-
74	TD Identification; TD Documentation/Representation; TD Communication; TD Repayment	Software architect, Dev. team	Software architect	Software architect	-	-	Software architect; Dev. team	-
83	TD Identification; TD Documentation/Representation; TD Communication; TD Measurement; TD Prioritization; TD Repayment	Dev. team	Dev. team	Dev. team	Dev. team	Dev. team	Dev. team	-
84	TD Identification; TD Documentation/Representation; TD Communication; TD Measurement; TD Prioritization	Team leader; Dev. team	Team leader; Dev. team	Team leader; Dev. team	-	Team leader	-	-

Participant ID	Technical debt management						
	Question 20						
	TD Identification	TD Documentation/ Representation	TD Communication	TD Measurement	TD Prioritization	TD Repayment	TD Prevention
20	-	-	Requirements/Business Analyst	-	Nothing to declare	Nothing to declare	-
24	-	-	-	-	-	-	Quality Analyst
26	Client support	Product	-	-	-	Development	-
29	-	Nothing to declare	Nothing to declare	-	Nothing to declare	-	Nothing to declare
52	Nothing to declare	-	-	-	-	-	-
74	Nothing to declare	Nothing to declare	Nothing to declare	-	-	Nothing to declare	-
83	Nothing to declare	Nothing to declare	Nothing to declare	Nothing to declare	Nothing to declare	Nothing to declare	-
84	Dev. team	Dev. team	Dev. team	Dev. team	Product Owner	-	-

Participant ID	Technical debt management							
	Question 21							
	TD Identification	TD Documentation/ Representation	TD Communication	TD Measurement	TD Prioritization	TD Repayment	TD Monitoring	TD Prevention
20	7	8	7	6	5	7	7	8
24	5	6	6	6	8	6	6	8
26	8	6	3	4	5	7	1	2
29	7	6	6	5	7	5	5	8
52	8	3	4	2	5	7	6	4
74	5	3	2	1	8	7	4	6
83	6	5	7	4	8	8	3	6
84	6	1	2	3	5	4	7	8

Participant ID	Technical debt identification						
	Question 22	Question 23	Question 24	Question 25	Question 26	Question 27	Question 28
26	Yes	Yes	Continuous identification	Manual code inspection	Yes	TD Type	Design Debt; Documentation Debt
52	No	-	Whenever there's a problem	Checklist	No	N/A	N/A
74	No	-	Whenever there's a problem	Manual code inspection; SonarQube/SQALE	No	N/A	N/A
83	Yes	No	Continuous identification	Manual code inspection; Dependency analysis; Checklist; SonarQube/SQALE; CheckStyle; Findbugs	Yes	Others: Effort to correct vs. Business value	N/A
84	Yes	Yes	Continuous identification	Manual code inspection; SonarQube/SQALE	Yes	Origin artifact	N/A

Participant ID	Technical debt documentation/representation				Technical debt communication		Technical debt measurement			
	Question 29	Question 30	Question 31	Question 32	Question 33	Question 34	Question 35	Question 36	Question 37	Question 38
20	-	-	-	N/A	Meetings with some stakeholders	Discussion forums; Others: GitLab	-	-	N/A	N/A
26	Yes	Yes	General backlog	TD list or backlog	-	N/A	-	-	N/A	N/A
29	Yes	No	General backlog	Others: Trello	Meetings with some stakeholders	Others: Trello	-	-	N/A	N/A
74	No	-	General backlog	TD list or backlog	Meetings with some stakeholders	Discussion forums	-	-	N/A	N/A
83	Yes	Yes	Specific backlog	TD list or backlog; Specific artifact(s) to debt documentation	Meetings with some stakeholders	Specific TD meetings	No	Metrics on specific information about the TD item	Others: Estimated effort vs. Value	Manual measurement; SonarQube
84	No	-	General backlog	JIRA	Informal discussion	Discussion forums	Yes	Simple information	Person-hours or person-months; LOC of the TD item	SonarQube; JIRA

Participant ID	Technical debt prioritization			Technical debt repayment		Technical debt prevention	
	Question 39	Question 40	Question 41	Question 42	Question 43	Question 46	Question 48
20	Simplified estimates	Issue classification	Most impact on the client; Most impact on the project	According to the moment's necessity	Refactoring; Code rewriting	-	N/A
24	-	N/A	N/A	-	N/A	No	Guidelines; Code patterns; Code reviews; Retrospective meetings; Definition of Done
26	-	N/A	N/A	According to the moment's necessity	Refactoring; Code rewriting	-	N/A
29	Simplified estimates	Issue classification	Most impact on the client; Poor cost/benefit relation	-	N/A	No	Guidelines; Code patterns; Definition of Done
74	-	N/A	N/A	Repay only when it is not possible to avoid any more	Refactoring; Redesign; Code rewriting	-	N/A
83	Simplified estimates	Specific cost/benefit analysis models	Most impact on the client; Most severity to the project; Most impact on the project; Items in used parts of the system	Continuous repayment planning	Code rewriting; Meetings/Workshops; Training	-	N/A
84	Others: Criticality	Issue classification	Most impact on the client; Most severity to the project; Most impact on the project	-	N/A	-	N/A

Appendix C – Information on the Studies from the Literature Review

This appendix presents some additional information on the 81 studies obtained from the literature review conducted to assess new technologies to manage the technical debt in software projects.

C.1 Introduction

In this appendix, we present additional data of the studies obtained from the literature review. For each paper, several data were extracted, as indicated on the extraction form included in Table 4.11. Those data were then synthesized in a spreadsheet, and the results were discussed in chapter 4. Aiming at better replicability of the study, the following tables present both the demographic data and the quality assessment results, broken down in each question, as presented in 4.4.6. Finally, this appendix presents a description of each technology.

C.2 Demographic data

ID	Paper	Year
S1	A benchmarking-based model for technical debt calculation	2014
S2	A case study in locating the architectural roots of technical debt	2015
S3	A contextualized vocabulary model for identifying technical debt on code comments	2015
S4	A debt-aware learning approach for resource adaptations in cloud elasticity management	2017
S5	A decision-support system approach to economics-driven modularity evaluation	2014
S6	A fluctuation-based modeling approach to quantification of the technical debt on mobile cloud-based service level	2015
S7	A formal approach to technical debt decision making	2013
S8	A framework for estimating interest on the technical debt by monitoring developer activity related to code comprehension	2014
S9	A game theoretic formulation of the technical debt management problem in cloud systems	2017
S10	A portfolio approach to technical debt management	2011
S11	A semi-automated framework for the identification and estimation of Architectural Technical Debt: A comparative case-study on the modularization of a software component	2018
S12	A strategy based on multiple decision criteria to support technical debt management	2017
S13	A threshold-based approach to technical debt	2012
S14	An empirical model of technical debt and interest	2011
S15	An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt	2016
S16	An open tool for assisting in technical debt management	2017
S17	Architectural debt management in value-oriented architecting	2014
S18	Architectural technical debt identification based on architecture decisions and change scenarios	2015

ID	Paper	Year
S19	Architecture viewpoints for documenting architectural technical debt	2016
S20	Assessing technical debt by identifying design flaws in software systems	2012
S21	CloudMTD: Using real options to manage technical debt in cloud-based service selection	2013
S22	DebtFlag: Technical debt management with a development environment integrated tool	2013
S23	Designite - A software design quality assessment tool	2016
S24	Developing processes to increase technical debt visibility and manageability - An action research study in industry	2016
S25	Estimating the breaking point for technical debt	2015
S26	Estimating the principal of an application's technical debt	2012
S27	Gamification: A game changer for managing technical debt? A design study	2018
S28	How “specification by example” and test-driven development help to avoid technical debt	2016
S29	Identifying and quantifying architectural debt	2016
S30	Identifying and visualizing architectural debt and its efficiency interest in the automotive domain: A case study	2015
S31	Identifying self-admitted technical debt in open source projects using text mining	2017
S32	Integrating technical debt management and software quality management processes: A normative framework and field tests	2017
S33	JSpIRIT: A flexible tool for the analysis of code smells	2015
S34	Lessons learned from the ProDebt research project on planning technical debt strategically	2017
S35	Managing technical debt with the SQALE method	2012
S36	Measuring and monitoring technical debt	2011
S37	Minimizing refactoring effort through prioritization of classes based on historical, architectural and code smell information	2016
S38	Minimizing technical debt: Developer's viewpoint	2012
S39	On the role of requirements in understanding and managing technical debt	2012
S40	On the value of a prioritization scheme for resolving the self-admitted technical debt	2018
S41	Practical technical debt discovery by matching patterns in assessment graph	2016
S42	A pragmatic approach for managing technical debt in legacy software project	2016
S43	Prioritizing design debt investment opportunities	2011
S44	Prioritizing technical debt in database normalization using portfolio theory and data quality metrics	2018
S45	Recommending when technical design debt should be self-admitted	2017
S46	Restraining technical debt when developing large-scale Ajax applications	2013
S47	Rework effort estimation of self-admitted technical debt	2016
S48	Searching for build debt: Experiences managing technical debt at Google	2012
S49	Selling the investment to pay down technical debt: The code Christmas tree	2011
S50	Software implementation knowledge management with technical debt and network analysis	2014
S51	SonarQube as a tool to identify software metrics and technical debt in the source code through static analysis	2017
S52	Startups and technical debt: Managing technical debt with visual thinking	2017
S53	Supporting technical debt cataloging with TD-tracker tool	2015
S54	Sustainability debt: A portfolio-based approach for evaluating sustainability requirements in architecture	2016
S55	Systematic elaboration of compliance requirements using compliance debt and portfolio theory	2014
S56	Technical debt management in the Brazilian federal administration	2015
S57	Technical debt management with genetic algorithms	2016
S58	Technical debt principal assessment through structural metrics	2017
S59	Technical debt prioritization using predictive analytics	2016
S60	Technical debt reduction using a game theoretic competitive source control approach	2015

ID	Paper	Year
S61	Technical debt reduction using search-based automated refactoring	2016
S62	Toward measuring defect debt and developing a recommender system for their prioritization	2015
S63	Towards a model for optimizing technical debt in software products	2013
S64	Towards a modularity-based technical debt prioritization approach	2016
S65	Towards a prioritization of code debt: A code smell intensity index	2015
S66	Towards a technical debt-management framework based on cost-benefit analysis	2015
S67	Towards an open-source tool for measuring and visualizing the interest of the technical debt	2015
S68	Towards proactive management of technical debt by software metrics	2015
S69	Towards triaging code-smell candidates via runtime scenarios and method-call dependencies	2017
S70	Understanding test debt	2017
S71	Using analytics to quantify interest in the self-admitted technical debt	2016
S72	Using natural language processing to detect self-admitted technical debt automatically	2017
S73	Using real options to manage technical debt in requirements engineering	2015
S74	VisMinerTD - An open source tool to support the monitoring of the technical debt evolution using software visualization	2015
S75	Visualizing architectural dependencies	2012
S76	Visualizing and managing technical debt in agile development	2013
S77	When-to-release decisions in consideration of technical debt	2014

C.3 Source and source type

ID	Source	Source Type
S1	International conference on quality software	Conference
S2	International conference on software engineering	Conference
S3	International workshop on managing technical debt	Workshop
S4	International conference on service-oriented computing	Conference
S5	Economics-driven software architecture	Book Chapter
S6	Globecom workshops	Workshop
S7	International ACM Sigsoft conference on quality of software architectures	Conference
S8	International workshop on managing technical debt	Workshop
S9	International conference on telecommunications	Conference
S10	International workshop on managing technical debt	Workshop
S11	Information and software technology	Journal
S12	Euromicro conference on software engineering and advanced applications	Conference
S13	ACM Sigsoft software engineering notes	Journal
S14	International workshop on managing technical debt	Workshop
S15	International conference on software engineering companion	Conference
S16	Euromicro conference on software engineering and advanced applications	Conference
S17	Economics-driven software architecture	Book Chapter
S18	Working IEEE/IFIP conference on software architecture (WICSA)	Conference
S19	Software quality assurance	Book Chapter
S20	IBM Journal of research and development	Journal
S21	International workshop on managing technical debt	Workshop
S22	International workshop on managing technical debt	Workshop
S23	International workshop on bringing architectural design thinking Into developers' daily activities	Workshop
S24	International conference product-focused software process improvement	Conference
S25	International workshop on managing technical debt	Workshop
S26	IEEE Software	Journal
S27	ArXiv preprint	Other
S28	International workshop on managing technical debt	Workshop
S29	International conference on software engineering	Conference
S30	International workshop on managing technical debt	Workshop
S31	Empirical software engineering	Journal
S32	IEEE Transactions on software engineering	Journal
S33	International Conference of the Chilean computer science society	Conference
S34	International conference on product-focused software process improvement	Conference
S35	IEEE Software	Journal
S36	Advances in computers	Book Chapter
S37	International workshop on technical debt analytics	Workshop
S38	International conference on software engineering and mobile application modeling and development	Conference
S39	International workshop on managing technical debt	Workshop
S40	Journal of systems and software	Journal
S41	International workshop on managing technical debt	Workshop
S42	India software engineering conference	Conference
S43	International workshop on managing technical debt	Workshop
S44	International workshop on managing technical debt	Conference
S45	Software maintenance and evolution	Conference
S46	International conference on building and exploring web-based environments	Conference
S47	International workshop on technical debt analytics	Workshop
S48	International workshop on managing technical debt	Workshop
S49	Agile Conference	Conference

ID	Source	Source Type
S50	International conference on research challenges in information science	Conference
S51	International workshop on computer science and engineering	Workshop
S52	International workshop on software engineering for startups	Workshop
S53	Advances in software engineering	Journal
S54	International conference on software engineering companion	Conference
S55	Working conference on requirements engineering: a foundation for software quality	Conference
S56	Brazilian workshop on agile methods	Workshop
S57	Euromicro conference on software engineering and advanced applications	Conference
S58	Euromicro conference on software engineering and advanced applications	Conference
S59	International conference on software engineering companion	Conference
S60	International conference on computer applications in industry and engineering	Conference
S61	Journal of systems and software	Journal
S62	International doctoral symposium on empirical software engineering	Symposium
S63	International workshop on managing technical debt	Workshop
S64	Full-scale software engineering/Current trends in release engineering	Other
S65	International workshop on managing technical debt	Workshop
S66	International conference on software engineering advances	Conference
S67	International workshop on managing technical debt	Workshop
S68	Symposium on programming languages and software tools	Symposium
S69	XP Workshops	Workshop
S70	Trends in software testing	Journal
S71	International workshop on technical debt analytics	Workshop
S72	IEEE Transactions on software engineering	Journal
S73	International requirements engineering conference	Conference
S74	International conference on enterprise systems	Conference
S75	International workshop on managing technical debt	Workshop
S76	International conference on Agile software development	Conference
S77	International workshop on managing technical debt	Workshop

C.4 Quality assessment

Quality assessment of papers									
ID	Q1	Q2	Q3	Q4	Q5	Q6	Q7	Q8	Sum
S1	1.0	0.5	0.5	0.5	0.5	1.0	0.0	0.6	4.6
S2	0.5	1.0	0.5	1.0	1.0	0.5	0.0	0.8	5.3
S3	1.0	0.5	1.0	1.0	1.0	0.0	1.0	0.6	6.1
S4	0.5	1.0	0.5	1.0	0.0	0.5	1.0	0.6	5.1
S5	1.0	0.5	1.0	1.0	0.0	0.0	0.0	0.8	4.3
S6	1.0	0.5	1.0	1.0	0.5	0.5	0.5	0.2	5.2
S7	1.0	0.5	0.5	1.0	1.0	1.0	0.5	0.2	5.7
S8	1.0	0.5	1.0	0.5	0.5	0.5	0.0	0.2	4.2
S9	1.0	1.0	0.5	0.5	0.5	0.5	0.0	0.2	4.2
S10	1.0	0.5	0.5	1.0	0.5	0.5	1.0	0.0	5.0
S11	1.0	1.0	1.0	0.5	0.5	1.0	1.0	0.8	6.8
S12	1.0	0.5	0.5	1.0	0.5	1.0	1.0	0.6	6.1
S13	1.0	0.5	0.5	0.5	0.0	1.0	1.0	0.8	5.3
S14	1.0	0.5	1.0	1.0	0.5	1.0	1.0	1.0	7.0
S15	1.0	1.0	1.0	0.5	0.5	0.5	1.0	0.8	6.3
S16	1.0	0.5	1.0	1.0	0.5	0.5	0.0	0.0	4.5
S17	1.0	1.0	1.0	0.5	0.5	0.5	0.0	0.8	5.3
S18	1.0	1.0	1.0	0.5	0.5	1.0	1.0	0.8	6.8
S19	1.0	1.0	1.0	0.5	0.5	0.5	1.0	0.8	6.3
S20	1.0	1.0	0.5	1.0	0.5	1.0	1.0	0.6	6.6
S21	1.0	1.0	1.0	0.5	0.5	0.5	0.0	0.2	4.7
S22	1.0	0.5	0.5	1.0	1.0	1.0	1.0	0.0	6.0
S23	1.0	1.0	0.5	1.0	0.5	0.5	0.5	0.6	5.6
S24	1.0	0.5	1.0	0.5	0.5	1.0	1.0	0.8	6.3
S25	1.0	0.5	0.5	1.0	0.5	0.5	1.0	0.2	5.2
S26	1.0	0.5	1.0	0.5	0.5	0.5	0.5	0.2	4.7
S27	1.0	0.0	0.5	0.5	1.0	0.5	0.5	0.4	4.4
S28	1.0	0.5	0.5	0.5	0.5	1.0	0.0	0.8	4.8
S29	1.0	1.0	1.0	0.5	0.5	0.5	1.0	0.2	5.7
S30	1.0	1.0	0.5	0.5	0.5	0.5	1.0	0.8	5.8
S31	1.0	0.5	1.0	0.0	0.5	0.5	1.0	0.6	5.1
S32	1.0	0.5	0.5	1.0	0.5	1.0	1.0	0.8	6.3
S33	1.0	1.0	1.0	1.0	0.5	1.0	0.0	0.2	5.7
S34	1.0	0.5	1.0	0.5	1.0	1.0	1.0	0.8	6.8
S35	1.0	0.5	1.0	1.0	0.5	0.0	1.0	0.2	5.2
S36	1.0	0.5	1.0	1.0	0.5	0.5	0.5	0.2	5.2
S37	1.0	0.5	1.0	1.0	0.5	0.5	1.0	0.2	5.7
S38	0.5	0.5	1.0	0.5	0.5	0.0	0.0	0.8	3.8
S39	1.0	0.5	0.5	0.5	0.5	0.5	0.0	0.0	3.5
S40	1.0	0.5	1.0	1.0	0.5	1.0	1.0	0.6	6.6
S41	1.0	0.5	1.0	1.0	1.0	0.5	0.0	0.6	5.6
S42	1.0	0.5	0.5	1.0	1.0	0.5	0.0	0.8	5.3
S43	1.0	1.0	1.0	0.5	0.5	0.5	0.0	0.2	4.7
S44	1.0	1.0	0.5	1.0	0.5	1.0	0.0	0.6	5.6
S45	1.0	1.0	0.5	0.5	0.5	0.5	0.0	0.6	4.6
S46	1.0	0.5	1.0	1.0	1.0	0.0	0.0	0.4	4.9
S47	1.0	0.5	0.5	0.5	0.5	1.0	1.0	0.6	5.6
S48	0.5	1.0	0.5	0.5	0.0	0.0	0.0	1.0	3.5
S49	0.5	0.5	0.5	0.5	0.5	1.0	1.0	1.0	5.5
S50	0.5	0.5	0.5	0.5	0.5	1.0	1.0	0.0	4.5
S51	1.0	0.5	0.5	0.5	0.0	1.0	0.0	0.6	4.1
S52	0.5	0.5	0.5	0.5	0.5	0.5	0.0	0.0	3.0
S53	1.0	1.0	1.0	1.0	0.5	0.5	0.0	0.2	5.2
S54	1.0	0.5	0.5	0.5	0.0	1.0	1.0	0.6	5.1
S55	1.0	1.0	0.5	0.5	0.0	1.0	1.0	0.2	5.2
S56	1.0	0.5	1.0	0.5	0.5	0.5	0.0	0.0	4.0
S57	1.0	0.5	0.5	0.5	1.0	1.0	0.0	0.6	5.1
S58	1.0	1.0	0.5	0.5	0.0	1.0	1.0	0.6	5.6
S59	1.0	0.5	1.0	0.5	0.0	1.0	0.0	0.0	4.0
S60	1.0	0.5	0.5	0.5	0.0	1.0	0.0	0.6	4.1
S61	1.0	0.5	0.5	0.5	0.0	1.0	1.0	0.6	5.1
S62	1.0	1.0	1.0	0.5	0.0	0.5	0.0	0.0	4.0
S63	1.0	0.5	0.5	0.0	0.0	1.0	0.0	0.0	3.0
S64	1.0	1.0	1.0	1.0	0.0	0.0	1.0	0.2	5.2
S65	1.0	0.5	0.5	0.5	0.0	1.0	1.0	0.6	5.1
S66	1.0	0.5	1.0	1.0	0.0	0.0	0.5	0.0	4.0
S67	1.0	0.5	1.0	0.5	0.0	1.0	1.0	0.2	5.2
S68	0.5	0.5	0.5	0.5	0.0	1.0	1.0	1.0	5.0
S69	1.0	0.5	0.5	0.5	0.0	0.5	0.5	0.2	3.7
S70	1.0	1.0	1.0	0.5	0.5	0.5	0.0	0.8	5.3
S71	1.0	1.0	1.0	0.5	0.0	0.5	0.5	0.2	4.7
S72	1.0	0.5	1.0	1.0	0.5	1.0	1.0	0.6	6.6
S73	1.0	1.0	0.0	0.5	0.5	1.0	1.0	0.2	5.2
S74	1.0	0.5	1.0	0.5	0.5	1.0	1.0	0.2	5.7
S75	1.0	1.0	0.5	0.0	0.5	1.0	0.0	0.8	4.8
S76	1.0	0.5	0.5	0.5	1.0	1.0	0.0	1.0	5.5
S77	1.0	0.5	1.0	0.5	0.5	1.0	1.0	0.2	5.7

C.5 Description of technologies

ID	Description
S1	Approach for Calculating Remediation Costs of TD on benchmarking-oriented quality assessments.
S2	Approach/Strategy to identify and quantify software architecture as a set of design rule spaces (DRSpaces).
S3	Contextualized Vocabulary Model to identify TD on code comments (CVM-TD). CVM-TD uses word classes and code tags to support TD identification.
S4	A technical debt-aware learning approach for autonomous elasticity management based on a reinforcement learning of debts in resource provisioning; the adaptation pursues strategic decisions that value the potential utility produced by the gaps between resource supply and demand.
S5	A decision-support system (DSS) approach to the modularity debt management. Using such a system, managers would be able to play out various “what-if” scenarios to make informed decisions regarding refactoring.
S6	A novel fluctuation-based quantification model and a quantification tool as a proof-of-concept, which is based on a cost-benefit appraisal, intending to quantify and evaluate the technical debt on mobile cloud-based service level, when fluctuations in demand occur.
S7	A detailed, formal analysis of decision making on technical debt in development. It considers the effective technical debt. This approach takes into account the cost of changes in each step evolution.
S8	A framework that continuously estimates interest payments using code comprehension metrics produced by a tool that monitors developer activities in the Integrated Development Environment.
S9	A game theoretic formulation of the technical debt management problem on the cloud-based service level. A technical debt measurement game is constructed, parameterizing the current number of players per service, while each new end-user can choose any of the offered cloud-based services.
S10	A portfolio approach that leverages the portfolio management theory in the finance domain to determine the optimal collection of technical debt items that should be incurred or held. The core component of the proposed approach is a “technical debt list.”
S11	A technique to identify Architectural Technical Debt in the form of a non-modularized component and to quantify the convenience of its repayment. It uses a holistic framework for the semi-automated identification and estimation of Architectural Technical Debt in the form of non-modularized components.
S12	A TDM strategy, named TD Manager that uses multiple decision criteria to evaluate debt items in order to facilitate decision making on their payment.
S13	An approach for establishing program specific thresholds to define manageable levels of technical debt.
S14	An approach to quantify debts and interest. The estimating taking accounts the rework fraction and rebuilds value.
S15	A method called AnaConDebt that aid architects and managers to understand if Architecture Technical Debt will generate a costly and growing interest to be paid (refactoring) or not.
S16	A tool (TEDMA tool) for monitoring technical debt over the software evolution and that it is open to integrate third-party tools. TEDMA is based on the analysis of source code repositories and is useful for research using empirical data extracted from software projects.
S17	A Decision-Based ATDM approach, which it is based on a conceptual model of ATD, a template for Documentation an ATD item and a process of ATDM.
S18	An ATD identification approach based on architecture decisions and change scenarios. The main goal is supporting for identification ATD based on ADs made during the architecting process and change scenarios for software maintenance and evolution.
S19	An approach based on a set of architecture viewpoints related to ATD (ATD viewpoints in short), which it aims to facilitate the documentation of ATD through adopting the architecture documentation approach mandated by ISO/IEC/IEEE 42010 (ISO/IEC/IEEE, 2011), which is based on architecture viewpoints.
S20	A framework for assessing technical debt by exposing debt symptoms at the design level.
S21	An option-based approach to inform the selection of candidate web services with varying

ID	Description
	costs, utilities and value they imply on the system.
S22	A tool for capturing, tracking and resolving technical debt in software projects. DebtFlag integrates into the development environment and provides developers with lightweight documentation tools to capture technical debt and link them to corresponding parts in the implementation.
S23	Designite is a software design quality assessment tool. It offers comprehensive support to detect a wide variety of design smells and computes various metrics at different granularities.
S24	A process for technical debt identification, documentation, and prioritization. The outcome of this study includes new processes to increase the visibility and manageability of technical debt, which can be used in the future for better decision-making.
S25	An approach for estimating the technical debt breaking point, based on a search-based optimization tool that is capable of identifying the distance of actual object-oriented design to the corresponding optimum one. It is based on the principal and interest values.
S26	A formula with adjustable parameters for estimating the principal of technical debt from structural quality data.
S27	A customized gamification tool, Themis that integrates with the project version control tool as well as with the SonarQube tool for identifying and measuring TD. Themis uses gamified features such as points, leaderboards, and challenges as a way to motivate developers and help managers with TDM.
S28	The use of "specification by example" methodology and test drive development to help to avoid the accumulation of technical debt.
S29	An approach for identifying and quantifying architectural debt, describing a novel history coupling probability matrix for this purpose, and identify architecture debts using four patterns of architectural flaws shown to correlate with reduced software quality.
S30	A visualization tool to provide a suitable representation of the ATD items and their interest previously elicited. The tool automatically collects the necessary data from models and databases, and compare the high-level architecture with the detailed design, calculating the technical debt and visualize it.
S31	A text mining-based approach to automatically detect self-admitted technical debt in source code comments.
S32	A normative process framework that systematically incorporates steps for managing technical debt in commercial software production.
S33	A flexible tool to identify and prioritize technical debt in the form of code smells. The tool is flexible to allow developers to add new smell detection strategies and to prioritize smells, and groups of smells, based on the configuration of their various criteria.
S34	The ProDebt approach aims at developing an innovative methodology and a software tool to support the strategic planning of technical debt in the context of agile software development.
S35	The SQALE method defines additional indexes and indicators to analyze and understand the debt. For this, the method organizes and groups requirements according to a specific chronology.
S36	A framework to support technical debt management.
S37	An approach for identifying and prioritizing object-oriented software classes in need of refactoring.
S38	Set of thirteen steps with practices for reducing TD.
S39	Requirements modeling tool, RE-KOMBINE. It represents technical debt using the notion of optimal solutions to a requirements problem.
S40	A framework which it is a set of steps - a prioritization scheme to provide a framework for minimizing self-admitted technical debt (SATD) based on identified textual indicators by Potdar and Shihab (2014). It is based on a text mining algorithm to mine indicative source code comments of SATD.
S41	Assessment Graph, a graph of artifacts relevant to a typical architectural assessment, formulating everyday assessment tasks as graph matching patterns, thus building TD discovery and assessment patters.
S42	Four stages of pragmatic approach to managing technical debt in legacy systems.
S43	An approach to using cost/benefit analysis to prioritize technical debt reduction work by ranking the value and interest of design debt caused by god classes. The method is based

ID	Description
	on metric analysis and software repository mining.
S44	Approach to minimize the negative impact of the debt tables on performance taking into consideration the likely growth rate of the table size and henceforth, the risk of interest accumulation.
S45	A machine learning approach named TEDIOUS (TEchnical Debt IdentificatiOn System), which leverages various kinds of method-level features as independent variables, including source code structural metrics, readability metrics and warnings raised by static analysis tools.
S46	The process to apply behavior-preserving transformations to the code in a way that the resulting code provides better reusability, compatibility among different components, and simplicity of the iterative software design process.
S47	A text mining technique for mining self-admitted technical debt tasks using source code comments.
S48	An approach to control and repay build debt.
S49	Visualization approach called code Christmas tree.
S50	An approach to applying knowledge management practices in the technical debt management, using previously presented tool, DebtFlag.
S51	Platform to manage and control the code quality in seven axes through the SQALE method.
S52	Duct taped technical debt, consisting of taking a duct tape roll and cutting a piece every time you issue technical debt.
S53	An approach to creating an integrated catalog of technical debts from different software development tasks. The approach is implemented by TD-Tracker tool, which can integrate different TD identification tools and import identified debts.
S54	An economics-driven architectural evaluation method which extends the Cost Benefits Analysis Method (CBAM) and integrates principles of modern portfolio theory to evaluate software architecture design decisions for sustainability.
S55	An economics-driven solution, which elaborates on the notion of obstacles handling in goal-oriented requirements engineering by using portfolio-based thinking and compliance debt analysis to manage compliance goals and their obstacles systematically.
S56	A model for the identification and measurement of the TD.
S57	An approach and tool to help organizations in decision making about developing new features or paying back known technical debt.
S58	Predictive model to quantify TD principal.
S59	A framework which makes use of a plethora of techniques ranging from data mining to prediction and decision models that project managers can use in their decision-making process to determine which technical debt is more critical and should be addressed first.
S60	A productivity game in the form of a competitive source control plug-in called the Build Game plug-in, which rewards technical debt-reducing actions.
S61	A specific strategy to combine refactoring techniques, search approaches and software metrics to reduce TD.
S62	An approach for measuring the principals and interests of defect debt.
S63	An optimization model that contrasts the patterns of technical debt accumulation in a software product with the patterns of consumer adoption of the product throughout its evolution.
S64	A prioritization approach for modularity debt.
S65	Intensity Index, to be used as an estimator to determine the most critical instances, prioritizing the examination of smells and, potentially, their removal.
S66	An approach called Cost-Benefit based Technical Debt Management (CoBeTDM). Its overall goal is to provide a framework to manage and reduces TD based on cost-benefit analysis for each release.
S67	MIND, an open-source tool to support the quantification and visualization of the interest.
S68	Specific software development process for TD management using metrics.
S69	Approach to obtain tailorable design documentation for object-oriented systems based on runtime tests, supporting a tool-supported triaging of code-smell candidates.
S70	Set of strategies for repaying test debt.
S71	An approach to use code metrics, in particular, the well-known Lines of Code (LOC) and Fan-In, to measure interest.
S72	An approach to automatically identify design and requirement self-admitted technical debt

ID	Description
	using Natural Language Processing (NLP). A Java-based tool reads from the database the data obtained by parsing the source code.
S73	A systematic method to apply the real options approach to manage the requirements of debt decisions.
S74	A tool, called VisMinerTD that aims to support developers in the identification and monitoring of TD in software projects.
S75	A modeling approach for visualizing dependency relationships as an extension of the current approaches, and supporting a general architectural knowledge capture, being better suited for heterogeneous environments.
S76	A technical debt board with main technical debt categories to manage and visualize the high-level debt, combined with tools to measure it at a low level (software metrics and another kind of static analysis).
S77	A methodology to effectively estimate technical debt, while considering debt in determining when-to-release product planning.

C.6 Additional information

Backgrounds for the selected technologies

Background	# of studies	Studies
Software metrics	29	S8; S11; S13; S14; S16; S20; S23; S26; S29; S30; S33; S34; S35; S37; S39; S40; S42; S43; S45; S51; S56; S58; S60; S64; S65; S67; S68; S71; S74
Software process	15	S2; S15; S17; S18; S24; S32; S48; S50; S53; S59; S66; S69; S70; S75; S77
Text mining	3	S3; S31; S47
Cost-benefit analysis	2	S6; S54
Portfolio management approach	3	S10; S44; S55
Real options	3	S5; S21; S73
Software development practices	3	S22; S28; S38
Games	2	S9; S27
Reinforcement learning	2	S4; S63
Search Based	2	S25; S61
Software design	2	S41; S46
Software quality assessment	2	S1; S36
Visualization Techniques	2	S52; S76
Architecture documentation	1	S19
Business Value	1	S63
Cost-based formalization	1	S7
Genetic algorithms	1	S57
Multicriteria decision	1	S12
Natural language processing	1	S72
Visual thinking	1	S52

Source artifacts for the selected studies

Project artifact	# of studies	Studies
Source code	47	S1; S2; S3; S5; S8; S11; S13; S14; S15; S16; S20; S22; S23; S25; S26; S29; S31; S33; S34; S35; S37; S40; S41; S42; S43; S45; S46; S47; S49; S50; S51; S52; S53; S56; S57; S58; S59; S60; S61; S63; S65; S68; S74; S71; S72; S74; S76
Any	13	S10; S12; S17; S18; S19; S21; S24; S27; S32; S36; S38; S66; S77
Others	7	S8; S10; S29; S31; S45; S46; S51; S73
Financial data on the project	2	S6; S73
Architecture models	2	S54; S75
Requirements elicitation	2	S39; S53
SaaS configuration attributes	1	S4
Software repositories	2	S62; S67
N/A	1	S63

Studies distribution over TDM activities

TDM activity	# of studies	Studies
Not specific	38	S13; S14; S17; S20; S21; S22; S23; S24; S25; S27; S29; S30; S32; S33; S34; S35; S36; S37; S39; S40; S42; S47; S48; S49; S50; S52; S53; S54; S55; S56; S62; S65; S66; S73; S74; S75; S76; S77
Measurement	43	S1; S2; S4; S5; S6; S7; S8; S9; S10; S11; S13; S14; S15; S16; S17; S20; S21; S22; S25; S26; S30; S30; S32; S33; S34; S35; S36; S37; S39; S40; S47; S48; S54; S56; S58; S61; S62; S65; S67; S78; S71; S75; S77
Identification	42	S2; S3; S5; S11; S12; S13; S14; S17; S18; S20; S21; S23; S24; S25; S30; S31; S32; S33; S34; S35; S36; S37; S39; S40; S41; S42; S45; S47; S48; S49; S50; S51; S53; S54; S55; S63; S65; S66; S69; S72; S74; S76
Prioritization	28	S5; S8; S9; S10; S12; S15; S17; S21; S24; S30; S32; S33; S35; S36; S37; S40; S42; S43; S44; S53; S57; S59; S62; S64; S65; S66; S73; S76
Monitoring	18	S4; S5; S9; S10; S13; S17; S20; S22; S27; S32; S36; S48; S49; S53; S66; S74; S76; S77
Prevention	14	S4; S6; S17; S23; S27; S28; S32; S34; S36; S38; S42; S55; S60; S75
Documentation/ Representation	15	S10; S17; S19; S22; S24; S30; S32; S36; S39; S42; S50; S52; S53; S75; S76
Repayment	11	S5; S11; S13; S17; S27; S32; S35; S36; S42; S46; S70
Communication	6	S22; S35; S36; S52; S75; S76

TD types for each study, divided by intentionality (as per McConnell (2007))

Dimension	# of studies	Studies
Intentional TD	14	S3; S4; S7; S15; S17; S21; S47; S52; S54; S55; S57; S62; S71; S73
Not specific	34	S6; S9; S10; S12; S13; S14; S18; S19; S20; S24; S25; S26; S27; S32; S34; S35; S36; S37; S38; S39; S42; S43; S44; S48; S53; S59; S60; S66; S69; S70; S74; S75; S76; S77
Unintentional TD	29	S1; S2; S5; S8; S11; S16; S22; S23; S28; S29; S30; S31; S32; S40; S41; S45; S46; S49; S50; S51; S56; S58; S61; S63; S64; S65; S67; S68; S72

TD types for each study, divided according to Alves et al.'s (2016) taxonomy

TD type	# of studies	Studies
Not specific	44	S3; S7; S10; S12; S13; S14; S16; S22; S24; S25; S26; S27; S28; S29; S31; S32; S33; S34; S35; S36; S37; S38; S40; S41; S42; S47; S49; S53; S54; S55; S57; S59; S60; S61; S63; S64; S65; S66; S68; S71; S72; S74; S76; S77
Code	50	S1; S3; S7; S8; S10; S12; S13; S14; S16; S22; S24; S25; S26; S27; S28; S29; S30; S32; S33; S34; S35; S36; S37; S38; S40; S41; S42; S46; S47; S49; S50; S51; S52; S53; S56; S57; S58; S59; S60; S61; S63; S64; S65; S66; S67; S68; S71; S74; S76; S77
Design	48	S3; S5; S7; S10; S12; S13; S14; S16; S20; S22; S23; S24; S25; S26; S27; S28; S29; S31; S32; S33; S34; S35; S36; S37; S38; S40; S41; S42; S43; S44; S45; S47; S49; S53; S56; S57; S59; S60; S61; S63; S64; S65; S69; S71; S72; S74; S76; S77
Architecture	44	S2; S3; S7; S10; S11; S12; S15; S16; S17; S18; S19; S22; S24; S24; S26; S27; S28; S29; S30; S31; S32; S33; S34; S35; S36; S37; S40; S40; S41; S42; S53; S54; S56; S57; S60; S63; S64; S66; S68; S71; S74; S75; S76; S77
Requirement	15	S7; S10; S12; S24; S32; S36; S39; S47; S54; S55; S57; S63; S72; S73; S76
Service	13	S4; S6; S7; S9; S10; S12; S21; S24; S23; S36; S57; S63; S76
Test	16	S7; S10; S12; S13; S24; S32; S36; S47; S49; S53; S57; S63; S68; S70; S74; S76
Documentation	13	S7; S10; S12; S24; S28; S32; S36; S47; S53; S57; S63; S72; S76
Defect	13	S8; S11; S13; S25; S33; S37; S49; S56; S60; S66; S67; S78; S80
Build	10	S7; S10; S12; S24; S32; S36; S50; S57; S63; S76
Versioning	10	S7; S10; S12; S24; S32; S36; S57; S63; S72; S76

Classification of studies according to the development model

Development model	# of studies	Studies
Independent of software process development	63	S1; S2; S3; S4; S5; S6; S8; S9; S10; S11; S12; S13; S14; S15; S17; S18; S10; S21; S22; S23; S24; S25; S26; S28; S29; S30; S31; S32; S33; S35; S36; S37; S38; S39; S40; S41; S42; S43; S45; S47; S48; S50; S51; S52; S53; S54; S54; S58; S59; S60; S61; S62; S63; S64; S65; S66; S67; S68; S69; S71; S72; S74; S75
Iterative/Agile software development	13	S7; S16; S20; S27; S34; S44; S46; S49; S56; S57; S73; S76; S77
Waterfall software development	1	S70

Classification of studies according to the system or project domain

System/Project domain	# of studies	Studies
Software development	9	S1, S3, S25; S29; S31; S40; S45; S68; S72
Different domains	6	S5; S26; S32; S34; S37, S41
Telecommunications	4	S11; S17; S18; S19
Cloud Applications	2	S4; S9
Mobile applications	2	S6; S42
Automotive	1	S30
Consulting	1	S12
Financial systems	1	S24
Government	1	S58
Health	1	S28
Oil & Gas	1	S77
Social networks	1	S21
N/A	47	S2; S7; S8; S10; S13; S14; S15; S16; S20; S22; S23; S27; S33; S35; S36; S38; S39; S43; S44; S46; S47; S48; S49; S50; S51; S52; S53; S54; S55; S57; S58; S59; S60; S61; S62; S63; S64; S65; S66; S67; S69; S70; S71; S73; S74; S75; S77

Distribution of the studies according to the programming language applicable to each selected technology

Programing language	# of studies	Studies
Not specific to a programming language	21	S13; S14; S35; S44; S46; S47; S48; S52; S54; S55; S56; S59; S62; S64; S68; S69; S73; S74; S75; S76; S77
Java	20	S1; S2; S3; S5; S20; S22; S25; S27; S33; S37; S41; S49; S50; S53; S58; S60; S61; S65; S71; S72
Different programming languages	3	S26; S51; S67
C	2	S11; S34
C++	2	S11; S70
C#	1	S43
N/A	28	S4; S6; S7; S8; S9; S10; S12; S15; S16; S17; S18; S19; S21; S24; S28; S29; S30; S31; S32; S36; S38; S39; S40; S42; S45; S57; S63; S66

Dependence on other technologies

Technology dependence	# of studies	Studies
Third-party tools	32	S1; S5; S14; S16; S22; S25; S26; S27; S29; S30; S34; S37; S38; S41; S42; S43; S45; S47; S50; S51; S53; S59; S60; S61; S65; S66; S67; S69; S71; S72; S76; S77
Proprietary tools	27	S3; S4; S6; S8; S9; S11; S12; S15; S16; S23; S25; S27; S29; S30; S31; S34; S39; S45; S46; S47; S48; S50; S53; S49; S65; S69; S72
Eclipse	5	S22; S31; S33; S50; S65
Software metrics	5	S13; S44; S58; S61; S68
Template	4	S10 S17; S18; S24
Extended third-party tools	4	S20; S31; S33; S59
Git	3	S60; S77; S74
Simulation tool	2	S4; S9
Specific artifacts	2	S19; S32
SQALE	2	S51; S56
Version control	2	S5; S50
Proprietary forms	1	S2
Proprietary spreadsheets	1	S7
Specific algorithms	1	S40
Tracking system	1	S5
N/A	15	S5; S21; S28; S35; S36; S52; S54; S55; S57; S62; S63; S64; S70; S73; S75

Distribution of selected studies over the type of evidence

Validation of the technology	# of studies	Studies
Toy example	22	S6; S7; S8; S9; S21; S25; S26; S29; S33; S35; S36; S37; S43; S53; S55; S64; S67; S69; S71; S73; S74; S77
Lab or classroom experiments	20	S1; S3; S4; S12; S20; S23; S31; S40; S41; S44; S45; S47; S51; S54; S57; S58; S60; S61; S65; S72
Case study	16	S2; S5; S11; S15; S17; S18; S19; S24; S28; S30; S32; S34; S38; S42; S70; S75
No evidence	11	S10; S16; S22; S39; S50; S52; S56; S59; S62; S63; S66
Industry practices	6	S13; S14; S48; S49; S68; S76
Specialist opinions	2	S27; S46

Appendix D – Evidence Briefings

This appendix presents the final version of the Evidence Briefings, whose design and development were detailed in chapter 5.

D.1 Introduction

In this appendix, we present the evidence briefings developed to aggregate the knowledge obtained in this dissertation, aiming to serve as a knowledge transfer medium between academic researches and practitioners from the software industry. Although part of this dissertation focused on the BSOs, the briefings were also developed in English so that it can be used by other researchers and practitioners from other countries. The English versions were also presented in chapter 5, from Figure 5.2 to Figure 5.4.

Additionally, this appendix includes the tables used in the briefings, to simplify the content visualization. The briefings were reduced to fit in the dissertation format.

13 steps for reducing TD
TDM activity: TD Prevention
TD types covered: Code TD, Architecture TD, Design TD
Source artifact: Any
Type of evidence: Case study
Reference: Krishna, V.; Basu, A. Minimizing Technical Debt: Developer's Viewpoint. International Conference on Software Engineering and Mobile Application Modelling and Development, Chennai, 2012.

Four stages of managing TD in legacy systems
TDM activity: TD Identification; TD Representation/Documentation; TD Prioritization; TD Repayment; TD Prevention
TD types covered: Code TD, Architecture TD, Design TD
Source artifact: Source code
Type of evidence: Case study
Reference: Gupta, R. K. et al. Pragmatic Approach for Managing Technical Debt in Legacy Software Project. 9th India Software Engineering Conference, Goa, 2016.

Approach to control and repay build debt
TDM activity: TD Identification; TD Monitoring; TD Measurement
TD types covered: Build TD
Source artifact: Specifications for building software
Type of evidence: Industry practices at Google
Reference: Morgenthaler, J. D. et al. Searching for build debt: Experiences managing technical debt at Google. 3rd International Workshop on Managing Technical Debt, Piscataway, 2012.

Proactive management of TD by software metrics
TDM activity: TD Measurement
TD types covered: Code TD, Architecture TD, Test TD
Source artifact: Source code
Type of evidence: Industry practices at Ericsson
Reference: Sandberg, A. B.; Staron, M.; Antinyan, V. Towards proactive management of technical debt by software metrics. 14th Symposium on Programming Languages and Software Tools, Tampere, 2015.

Strategies for repaying test debt
TDM activity: TD Repayment
TD types covered: Test TD
Source artifact: Source code
Type of evidence: Case study
Reference: Samarthyam, G.; Muralidharan, M.; Anna, R. K. Understanding Test Debt. Trends in Software Testing, Singapore, 2017

TD board to manage and visualize high-level debt
TDM activity: TD Identification; TD Representation/Documentation; TD Monitoring; TD Prioritization; TD Communication
TD types covered: Not specific to a TD type
Source artifact: Source code
Type of evidence: Industry practices at Petrobras
Reference: dos Santos, P. S. M. et al. Visualizing and managing technical debt in agile development: An experience report. International Conference on Agile Software Development, Berlin, 2013.

Technology	Description	Link	TD type	TDM activity	Source artifact	Evidence	Reference
JSpIRIT	A tool to identify and prioritize technical debt in the form of code smells.	goo.gl/dKnqvp	Code TD Architecture TD Design TD	TD Identification; TD Measurement; TD Prioritization	Source code	CS	[1]
SQALE plugin for SonarQube	A tool to analyze, measure, visualize and prioritize TD based on the SQALE quality model.	goo.gl/3oiAys	Code TD Architecture TD	TD Identification; TD Measurement; TD Prioritization; TD Communication	Source code	SP	[2]
SonarQube	An open platform for managing code quality.	goo.gl/6X2KGV	Code TD	TD Identification; TD Measurement; TD Monitoring	Source code	SP	[2]
CheckStyle	A tool to check Java code against coding standards.	goo.gl/iRFBTr	Code TD	TD Identification	Source code	SP	[2]
FindBugs	A tool to identify TD using automatic static analysis.	goo.gl/UFssPz	Code TD	TD Identification	Source code	SP	[2]
JIRA	A tool that allows task monitoring and management.	goo.gl/ga4zMv	Any	TD Representation/Documentation; TD Measurement	N/A	SP	[2]
Trello	A tool that allows task monitoring and management.	goo.gl/iBF6oA	Any	TD Representation/Documentation; TD Communication	N/A	SP	[2]
GitLab	A software repository manager.	goo.gl/kHsxMi	Any	TD Communication	N/A	SP	[2]

CS - Case study
SP - Survey with practitioners

Technology	Description	TD type	TDM activity	Source artifact	Evidence	Reference
CVM-TD	Model to identify TD on code comments.	Code TD; Architecture TD Design TD	TD Identification	Source code	IP	[3]
Not named	A decision-support system approach to the modularity debt management.	Design TD	TD Identification; TD Monitoring; TD Measurement; TD Prioritization; TD Repayment	Source code	EE	[4]
Not named	An approach to defining manageable levels of technical debt.	Code TD; Design TD Test TD	TD Identification; TD Monitoring; TD Measurement; TD Repayment	Source code	IP	[5]
Not named	An approach to quantifying TD.	Code TD; Design TD	TD Identification; TD Measurement	Source code	IP	[6]
AnaConDebt	A method that aid architects and managers to understand and quantify interest in architecture TD.	Architecture TD	TD Measurement; TD Prioritization	Source code	EE	[7]
Not named	A decision-based approach using a conceptual model of architecture TD.	Architecture TD	Not specific to a TDM activity	Any	EE	[8]
Not named	An identification approach based on architecture decisions and change scenarios.	Architecture TD	TD Identification	Any	EE	[9]
ATD viewpoints	An approach based on a set of architecture viewpoints related to architecture TD.	Architecture TD	TD Representation/Documentation	Any	EE	[10]
Not named	A process for TD identification, documentation, and prioritization.	Not specific to a TD type	TD Identification; TD Prioritization; TD Representation/Documentation	Any	EE	[11]
Not named	Methodology to help to avoid the accumulation of technical debt.	Code TD; Architecture TD Design TD Documentation TD	TD Prevention	Test cases	EE	[12]
Not named	A normative process framework for managing technical debt in commercial software production.	Not specific to a TD type	Not specific to a TDM activity	Any	EE	[13]
SQALE	The method that defines additional indexes and indicators to analyze and understand TD.	Code TD Architecture TD Design TD	TD Identification; TD Measurement; TD Prioritization; TD Repayment; TD Communication	Source code	EE	[14]
TD Template	A framework to support technical debt management.	Not specific to a TD type	Not specific to a TDM activity	Any	EE	[15]
Not named	Visualization approach.	Code TD; Design TD Test TD	TD Identification; TD Monitoring	Source code	IP	[16]
Duct taped TD	Visualization technique.	Code TD	TD Representation/Documentation; TD Communication	Source code	IP	[17]
CoBeTDM	A framework to manage and reduce TD.	Code TD; Architecture TD	TD Identification; TD Monitoring; TD Prioritization	Any	IP	[18]

EE - Experimentally evaluated
IP - Industry practice

DÍVIDA TÉCNICA: DEFINIÇÃO E CONCEITOS

Este *evidence briefing* apresenta informações sobre a definição e os principais conceitos de dívida técnica.

CONCEITOS

O que é Dívida Técnica?
Dívida Técnica (DT) é um conceito cunhado inicialmente por Ward Cunningham em 1992, mas desde então tem recebido diversas atualizações. Ela foi adotada por profissionais de métodos ágeis, mas possui uma ampla aplicação em todos os domínios de software. A principal definição de DT é [1]:

"Em sistemas intensivos de software, dívida técnica é uma coleção de construtos de design ou de implementação que são efetivos no curto prazo, mas que criam um contexto técnica que torna mudanças futuras mais caras ou impossíveis. A dívida técnica apresenta um risco real ou de contingência, cujo impacto é limitado a qualidades internas do sistema, principalmente manutenibilidade e capacidade de evolução."

Quatro aspectos precisam ser observados:

- **A DT não incorre apenas em código fonte**. É possível identificar DT na elicitação de requisitos ou em casos de teste, por exemplo;
- Para ser considerado DT, o problema deve causar um **benefício de curto prazo, em troca de um potencial custo futuro**;
- DT só é associada a atributos de qualidade interna, como manutenibilidade, então **defeitos não são DT!** É possível existir, no entanto, um tipo de DT de defeito (veja mais a seguir);
- O correto é **dívida técnica** e não "débito técnico"! A metáfora está associada à ideia de um "empréstimo" feito pelo profissional, em troca de "pagamentos" futuros, que ficam mais caros quanto maior for o tempo para quitação da dívida.

Então, o que deve ser considerado DT [2][3]?
- Código mal escrito, que viole regras;
- "Atalhos" tomados durante o design;
- Defeitos conhecidos, cuja eliminação é adiada para *sprints* ou ciclos de desenvolvimento futuros;
- Problemas arquiteturais, como violação de modularidade;
- *Code smells*;
- Aspectos gerais de qualidade interna, que afetem a manutenibilidade e a capacidade de evolução.

E o que não deve ser considerado DT [3]
- Defeitos;
- Problemas triviais de qualidade de código, que não violem as regras de código;
- Falta de processos de suporte;
- Funcionalidades não implementadas.

Quais outros conceitos são associados com DT?
Como DT é uma metáfora financeira, relacionada com uma dívida adquirida por alguém para obter um ganho de curto prazo, alguns outros conceitos de finanças são frequentemente associados com DT, como:
- **Principal:** O esforço que é exigido para resolver a diferença entre o nível de qualidade atual e ótimo, em um artefato de software imaturo ou no sistema de software completo;
- **Juros:** O esforço adicional requerido para ser investido na manutenção do software, devido ao decaimento da sua qualidade;
- **Pagamento:** O esforço gasto na melhoria da qualidade do software. Este esforço diminuirá o esforço necessário para tarefas de manutenção futuras.

Como DT pode ser classificada?
A classificação mais simples de itens de DT pode ser pela sua intenção:
- **DT intencional** é aquela causada por decisões estratégicas e planejadas, quando o time ou a organização decidem obter um ganho de curto prazo ao custo de um esforço de longo prazo. Por exemplo, a decisão de desenvolver uma solução simplificada de arquitetura, sabendo que ela pode não atender as necessidades futuras do projeto;
- **DT não-intencional** não incorre com um propósito estratégico, e usualmente aparece em um projeto devido à imaturidade ou falta de conhecimento dos profissionais. Por exemplo, um código mal escrito criado por um programador inexperiente pode ser um item de DT.

Outra classificação possível de DT é pelo seu tipo, ou seja, pelo que causa aquele item de DT específico. Alves et al [2] reuniu 15 diferentes tipos de DT. Os mais recorrentes em projetos de software estão listados abaixo:
- **Dívida de design:** Associada principalmente com violações dos princípios de um bom design orientado a objetos, como classes muito acopladas;
- **Dívida de arquitetura:** Refere-se a problemas relacionados com a arquitetura do software, como violação de modularidade;
- **Dívida de documentação:** Dívida relacionada a problemas observados na documentação do software;
- **Dívida de teste:** Dívida encontrada em atividades de teste, como casos de teste planejados que não foram executados;
- **Dívida de código:** Associada com problemas encontrados no código-fonte, que podem tornar a manutenção mais difícil, usualmente relacionada com más práticas de programação;
- **Dívida de construção (build):** Refere-se a problemas que podem prejudicar a tarefa de build, gerando um consumo desnecessário de tempo.

Para quem é este briefing?

Profissionais de engenharia de software que quiserem tomar decisões sobre problemas de qualidade interna e aplicar conhecimento científico ao gerenciar a dívida técnica.

De onde vêm as informações?

As informações neste *briefing* vêm de evidência coletada pelo autor através de uma revisão na literatura em diversas publicações, incluindo:
- [1] Avgeriou, P. et al. Managing Technical Debt in Software Engineering. In Dagstuhl Reports, 2016;
- [2] Alves, N. S. et al. Identification and management of technical debt: A systematic mapping study. Information and Software Technology, 2016;
- [3] Li, Z. et al. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 2015.

Para informações adicionais sobre o Grupo de Engenharia de Software Experimental na COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

Para informações adicionais sobre o Observatório DELFOS:

http://www.delfos.cos.ufrj.br

REFERÊNCIA DE PESQUISA
Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Dissertação de mestrado, Universidade Federal do Rio de Janeiro, 2018.

GERENCIAMENTO DA DÍVIDA TÉCNICA: ATIVIDADES E PRÁTICAS

Este *evidence briefing* apresenta informação sobre diferentes atividades para se gerenciar a dívida técnica, além de práticas obtidas através de um *survey* com profissionais e uma revisão da literatura.

EVIDÊNCIAS

Quais são as atividades de gerenciamento da dívida técnica?
Através de um estudo de mapeamento sistemático, Li et al [1] agruparam as atividades de gerenciamento da dívida técnica (GDT) em oito grupos, descritos abaixo:

- **Identificação da Dívida Técnica (DT):** Detecta DT causada por decisões técnicas de software, sejam intencionais ou não;
- **Medição da DT:** Avalia a relação custo/benefício de itens de DT conhecidos no software ou estima o valor geral de DT;
- **Priorização da DT:** Adota regras predefinidas para ranquear itens de DT, para suportar o processo de tomada de decisão;
- **Prevenção da DT:** Estabelece práticas para evitar DT potencial de ocorrer;
- **Monitoramento da DT:** Observa a evolução de itens de DT conhecidos ao longo do tempo;
- **Pagamento da DT:** Elimina ou reduz o impacto da DT (principal e juros) em um sistema de software;
- **Representação/documentação da DT:** Representa e codifica a DT em um padrão predefinido, para atender a preocupações das partes interessadas;
- **Comunicação da DT:** Divulga os itens de DT identificados para as partes interessadas.

Quais são as práticas para se gerenciar a DT?
Os *guidelines* ou práticas a seguir foram coletados em um *survey* com profissionais da indústria de software Brasileira (nenhum participante do *survey* informou práticas relacionadas ao monitoramento da DT)

- **Identificação da DT:** inspeção manual de código, análise de dependência, *checklist* de DT
- **Representação/documentação da DT:** Backlog de DT;
- **Comunicação da DT:** Fóruns de discussão, reuniões sobre DT;
- **Priorização da DT:** Análise de custo/benefício, classificação de *issues*;
- **Pagamento da DT:** Refatoração, redesign, reescrita de código;
- **Prevenção da DT:** Padrões de codificação, revisão de código, reuniões de retrospectiva, Definição de Pronto.

Os *guidelines* ou práticas a seguir foram coletados em relatos de experiência ou estudos de caso com a indústria, através de uma revisão da literatura:

13 passos para reduzir a DT
Atividade de GDT: Prevenção da DT
Tipos de DT cobertos: Dívida de código, Dívida de arquitetura, Dívida de design
Artefatos de origem: Qualquer
Tipo de evidência: Estudo de caso
Referência: Krishna, V.; Basu, A. Minimizing Technical Debt: Developer's Viewpoint. International Conference on Software Engineering and Mobile Application Modelling and Development, Chennai, 2012.
4 etapas para gerenciar a DT em sistemas legado
Atividade de GDT: Identificação da DT; Representação/Documentação da DT; Priorização da DT; Pagamento da DT; Prevenção da DT
Tipos de DT cobertos: Dívida de código, Dívida de arquitetura, Dívida de design
Artefatos de origem: Código-fonte
Tipo de evidência: Estudo de caso
Referência: Gupta, R. K. et al. Pragmatic Approach for Managing Technical Debt in Legacy Software Project. 9th India Software Engineering Conference, Goa, 2016.
Abordagem para controlar e pagar dívida de construção
Atividade de GDT: Identificação da DT; Monitoramento da DT; Medição da DT
Tipos de DT cobertos: Dívida de construção
Artefatos de origem: Specifications for building software
Tipo de evidência: Práticas da indústria no Google
Referência: Morgenthaler, J. D. et al. Searching for build debt: Experiences managing technical debt at Google. 3rd International Workshop on Managing Technical Debt, Piscataway, 2012.
Gerenciamento proativo da DT através de métricas de software
Atividade de GDT: Medição da DT
Tipos de DT cobertos: Dívida de código; Dívida de arquitetura; Dívida de teste
Artefatos de origem: Código-fonte
Tipo de evidência: Práticas da indústria na Ericsson
Referência: Sandberg, A. B.; Staron, M.; Antinyan, V. Towards proactive management of technical debt by software metrics. 14th Symposium on Programming Languages and Software Tools, Tampere, 2015.
Estratégias para pagar dívida de teste
Atividade de GDT: Pagamento da DT
Tipos de DT cobertos: Dívida de teste
Artefatos de origem: Código-fonte
Tipo de evidência: Estudo de caso
Referência: Samarthyam, G.; Muralidharan, M.; Anna, R. K. Understanding Test Debt. Trends in Software Testing, Singapore, 2017
Quadro de DT para gerenciar e visualizar dívidas de alto nível
Atividade de GDT: Identificação da DT; Representação/Documentação da DT; Monitoramento da DT; Priorização da DT; Comunicação da DT
Tipos de DT cobertos: Não específico para um tipo de DT
Artefatos de origem: Código-fonte
Tipo de evidência: Práticas da indústria na Petrobras
Referência: dos Santos, P. S. M. et al. Visualizing and managing technical debt in agile development: An experience report. International Conference on Agile Software Development, Berlin, 2013.

Para quem é este briefing?

Profissionais de engenharia de software que quiserem tomar decisões sobre problemas de qualidade interna e aplicar conhecimento científico ao gerenciar a dívida técnica.

De onde vêm as informações?

As informações neste *briefing* vêm de evidências coletadas pelo autor em uma revisão da literatura e em um *survey* com profissionais de software na indústria brasileira. As atividades de gerenciamento da dívida técnica são descritas de acordo com o estudo de mapeamento listado abaixo:

- [1] Li, Z. et al. A systematic mapping study on technical debt and its management. Journal of Systems and Software, 2015.

Para informações adicionais sobre o Grupo de Engenharia de Software Experimental na COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

Para informações adicionais sobre o Observatório DELFOS:

http://www.delfos.cos.ufrj.br

REFERÊNCIAS DE PESQUISA

Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Dissertação de mestrado, Universidade Federal do Rio de Janeiro, 2018.

Silva, V. M.; Jeronimo, H.; Travassos, G. H. ***A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil***. XXI Ibero-American Conference on Software Engineering, Bogotá, 2018.

GERENCIAMENTO DA DÍVIDA TÉCNICA: FERRAMENTAS E ESTRATÉGIAS

Este *evidence briefing* apresenta uma lista de ferramentas, metodologias, estratégias, abordagens ou *frameworks* para executar as diferentes atividades de gerenciamento da Dívida Técnica (DT).

Tecnologia	Descrição	Link	Tipo de DT	Atividade de GDT	Artefato de origem	Evidência	Referência
JSpIRIT	Uma ferramenta para identificar e priorizar a dívida técnica na forma de *code smells*.	goo.gl/dKnqvp	Código Arquitetura Design	Identificação; Medição Priorização	Código-fonte	EC	[1]
SQALE Plugin para SonarQube	Uma ferramenta para analisar, medir, visualizar e priorizar DT baseada no modelo de qualidade SQALE.	goo.gl/3oiAys	Código Arquitetura	Identificação; Medição Priorização; Comunicação	Código-fonte	SP	[2]
SonarQube	Uma plataforma aberta para gerenciar qualidade de código.	goo.gl/6X2KGV	Código	Identificação; Medição Monitoramento	Código-fonte	SP	[2]
CheckStyle	Uma ferramenta para verificar código Java com base em padrões de código.	goo.gl/iRFBTr	Código	Identificação	Código-fonte	SP	[2]
FindBugs	Uma ferramenta para identificar DT usando análise estática automática.	goo.gl/UFssPz	Código	Identificação	Código-fonte	SP	[2]
JIRA	Uma ferramenta que permite monitorar e gerenciar tarefas.	goo.gl/ga4zMv	Qualquer	Representação/Documentação; Medição	N/A	SP	[2]
Trello	Uma ferramenta que permite monitorar e gerenciar tarefas.	goo.gl/iBF6oA	Qualquer	Representação/Documentação; Comunicação	N/A	SP	[2]
GitLab	Um gerenciador de repositório de software.	goo.gl/kHsxMi	Qualquer	Comunicação	N/A	SP	[2]

EC - Estudo de caso

SP - *Survey* com profissionais

Tecnologia	Descrição	Tipo de DT	Atividade de GDT	Artefato de origem	Evidência	Referência
CVM-TD	Modelo para identificar a DT em comentários de código.	Código Arquitetura Design	Identificação	Código-fonte	PI	[3]
Não nomeada	Uma abordagem de sistema de suporte a decisões para o gerenciamento da dívida de modularidade.	Design	Identificação; Monitoramento; Medição; Priorização; Pagamento	Código-fonte	AE	[4]
Não nomeada	Uma abordagem para definir níveis gerenciáveis de dívida técnica.	Código Design Teste	Identificação; Monitoramento; Medição; Pagamento	Código-fonte	PI	[5]
Não nomeada	Uma abordagem para quantificar dívida técnica.	Código Design	Identificação; Medição	Código-fonte	PI	[6]
AnaConDebt	Um método para auxiliar arquitetos e gerentes a entender e quantificar juros em dívidas de arquitetura.	Arquitetura	Medição; Priorização	Código-fonte	AE	[7]
Não nomeada	Uma abordagem baseada em decisões usando um modelo conceitual de dívida de arquitetura.	Arquitetura	Não específico para uma atividade de GDT	Qualquer	AE	[8]
Não nomeada	Uma abordagem de identificação baseada em decisões de arquitetura e cenários de mudança.	Arquitetura	Identificação	Qualquer	AE	[9]
ATD viewpoints	Uma abordagem baseada em um conjunto de *viewpoints* de arquitetura relacionados a dívida de arquitetura.	Arquitetura	Representação/Documentação	Qualquer	AE	[10]
Não nomeada	Um processo para identificação, documentação e priorização de DT.	Não específica para um tipo de DT	Identificação; Priorização; Representação/Documentação	Qualquer	AE	[11]
Não nomeada	Metodologia para auxiliar a evitar o acúmulo de DT.	Código Arquitetura Design Documentação	Prevenção	Casos de teste	AE	[12]
Não nomeada	Um *framework* para GDT na produção de software comercial.	Não específica para um tipo de DT	Não específico para uma atividade de GDT	Qualquer	AE	[13]
SQALE	Método que define índices e indicadores para analisar e entender DT.	Código Arquitetura Design	Identificação; Medição Priorização; Pagamento Comunicação	Código-fonte	AE	[14]
TD Template	Um framework para suportar GDT.	Não específica para um tipo de DT	Não específico para uma atividade de GDT	Qualquer	AE	[15]
Não nomeada	Abordagem de visualização	Código Design Teste	Identificação; Monitoramento	Código-fonte	PI	[16]
Duct taped TD	Técnica de visualização	Código	Representação/Documentação Comunicação	Código-fonte	PI	[17]
CoBeTDM	Um *framework* para gerenciar e reduzir DT.	Código Arquitetura	Identificação; Monitoramento Priorização	Qualquer	PI	[18]

AE - Avaliado experimentalmente

PI - Prática da indústria

Para quem é este briefing?

Profissionais de engenharia de software que quiserem tomar decisões sobre problemas de qualidade interna e aplicar conhecimento científico ao gerenciar a dívida técnica.

De onde vêm as informações?

- [1] Vidal, S. et al. Identifying Architectural Problems through Prioritization of Code Smells. SBCARS, 2016;
- [2] Silva, V. M. et al. A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil. CibSE, 2018;
- [3] de Freitas Farias, M. A. et al. Investigating the Use of a Contextualized Vocabulary in the Identification of Technical Debt: A Controlled Experiment. ICEIS, 2016;
- [4] Cai, Y. et al. A decision-support system approach to economics-driven modularity evaluation. Economics-Driven Software Architecture, 2014;
- [5] Eisenberg, R. J. A threshold based approach to technical debt. Software Engineering Notes, 2012;
- [6] Nugroho, A. et al. An empirical model of technical debt and interest. Workshop on Managing Technical Debt, 2011;
- [7] Martini, A. et al. An empirically developed method to aid decisions on architectural technical debt refactoring: AnaConDebt. ICSE-C, 2016;
- [8] Li, Z. et al. Architectural debt management in value-oriented architecting. Economics-Driven Software Architecture, 2014;
- [9] Li, Z. et al. Architectural technical debt identification based on architecture decisions and change scenarios. WICSA, 2015;
- [10] Li, Z. et al. Architecture viewpoints for documenting architectural technical debt. Software Quality Assurance, 2016;
- [11] Yli-Huumo, J. et al. Developing Processes to Increase Technical Debt Visibility and Manageability – An Action Research Study in Industry. PROFES, 2016;
- [12] Trumler, W. et al. How "Specification by Example" and Test-Driven Development Help to Avoid Technical Debt. Workshop on Managing Technical Debt, 2016;
- [13] Ramasubbu, N. et al. Integrating Technical Debt Management and Software Quality Management Processes: A Normative Framework and Field Tests. IEEE Transactions on Software Engineering, 2017;
- [14] Letouzey, J. L. et al. Managing technical debt with the sqale method. IEEE software, 2012;
- [15] Seaman, C. et al. Measuring and monitoring technical debt. Advances in Computers, 2011;
- [16] Kaiser, M. et al. Selling the Investment to Pay Down Technical Debt: The Code Christmas Tree. AGILE, 2011;
- [17] Chicote, M. Startups and technical debt: managing technical debt with visual thinking. International Workshop on Software Engineering for Startups, 2017;
- [18] Harun, M. F. et al. Towards a technical debt-management framework based on cost-benefit analysis. ICSEA, 2015.

Para informações adicionais sobre o Grupo de Engenharia de Software Experimental na COPPE/UFRJ:

http://lens-ese.cos.ufrj.br/ese/

Para informações adicionais sobre o Observatório DELFOS:

http://www.delfos.cos.ufrj.br

REFERÊNCIAS DE PESQUISA

Silva, V. M. ***Technical Debt and its Management: Analysis and Application in the Brazilian Software Industry***. Dissertação de mestrado, Universidade Federal do Rio de Janeiro, 2018.

Silva, V. M.; Jeronimo, H.; Travassos, G. H. ***A Taste of the Software Industry Perception of Technical Debt and its Management in Brazil***. XXI Ibero-American Conference on Software Engineering, Bogotá, 2018.