Journal of Software Engineering Research and Development, 2023, 11:3, doi: 10.5753/jserd.2023.2581

Investigating the Relationship between Technical Debt Management and Software Development Issues

Clara Berenguer [Salvador University | claraberenguerledo@gmail.com]
Adriano Borges [Salvador University | arborges.12@gmail.com]
Sávio Freire [Federal Institute of Ceará and Federal University of Bahia | savio.freire@ifce.edu.br]
Nicolli Rios [Federal University of Rio de Janeiro | nicolli@cos.ufrj.br]
Robert Ramač [University of Novi Sad | ramac.robert@uns.ac.rs]
Nebojša Taušan [University of Novi Sad | nebojsa.tausan@ef.uns.ac.rs]
Boris Pérez [Francisco de Paula Santander University | br.perez41@uniandes.edu.co]
Camilo Castellanos [University of Los Andes | cc.castellanos87@uniandes.edu.co]
Darío Correal [University of Los Andes | dcorreal@uniandes.edu.co]
Alexia Pacheco [University of Costa Rica | alexia.pacheco@ucr.ac.cr]
Gustavo López [University of Costa Rica | gustavo.lopezherrera@ucr.ac.cr]
Manoel Mendonça [Federal University of Bahia | manoel.mendonca@ufba.br]
Davide Falessi [University of Rome Tor Vergata | d.falessi@gmail.com]
Carolyn Seaman [University of Maryland Baltimore County | cseaman@umbc.edu]
Vladimir Mandić [University of Novi Sad | vladman@uns.ac.rs]
Clemente Izurieta [Montana State University and Idaho National Laboratories | clemente.izurieta@montana.edu]
Rodrigo Spínola [Virginia Commonwealth University and Salvador University | spinolaro@vcu.edu]

Abstract

Context: The presence of technical debt (TD) brings risks to software projects. Managers must continuously find a cost-benefit balance between the benefits of incurring in TD and the costs of its presence in a software project. Much attention has been given to TD related to coding issues, but other types of debt can also have impactful consequences on projects. *Aims*: This paper seeks to elaborate on the growing need to expand TD research to other areas of software development, by analyzing six elements related to TD management, namely: causes, effects, preventive practices, reasons for non-prevention, repayment practices, and reasons for non-repayment of TD. *Method*: We survey and analyze, quantitatively and qualitatively, the answers of 653 software industry practitioners on TD to investigate how the previously mentioned elements are related to coding and non-coding issues of the software development process. *Results*: Coding issues are commonly related to the investigated elements but, indeed, they are only part of the TD Management stage. Issues related to the project planning and management, human factors, knowledge, quality, process, requirements, verification, validation, and test, design, architecture, and the organization are also common sources of TD. We organize the results in a hump diagram and specialize it considering the point of view of practitioners that have used agile, hybrid, and traditional process models in their projects. *Conclusion*: The hump diagram, in combination with the detailed results, provides guidance on what to expect from the presence of TD and how to react to it considering several issues of software development. The results shed light on TD management of software elements, beyond source code related artifacts.

Keywords: *Technical Debt, Technical Debt Management, Causes of Technical Debt, Effects of Technical Debt, Process Model*

1 Introduction

Technical debt (TD) refers to postponed tasks or immature artifacts in software projects that can bring short-term benefits (e.g., higher productivity and lower costs), but may have harmful impacts in the long run (Izurieta *et al*. 2012). By managing TD items, software teams can reduce the risks associated with these items, such as unexpected delays in system evolution or difficulty in achieving quality criteria defined for the project (Rios *et al*. 2020).

Technical debt management (TDM) is a challenging endeavor. Successful TDM is about reaching a balance between the benefits of incurring in TD and the later impacts of its presence in a software project (Lim *et al*. 2012, Guo *et al*. 2016). TDM must seek to define preventive practices to avoid potential TD items and the appropriate actions to repay incurring debt (Li *et al*. 2015, Ribeiro *et al*. 2016, Freire *et al*. 2020a, Freire *et al*. 2020b). TDM requires knowledge of the causes that lead software teams to incur debt items and the effects of their presence in software projects (Rios *et al*. 2020, Besker *et al*. 2020). Knowing the causes of TD can support software teams in understanding their project context and define preventive practices to avoid the debt. Having information on TD effects can aid in the prioritization of TD items to be paid off, supporting a more precise impact analysis and the identification of corrective actions to minimize possible negative consequences of TD items for the project.

Although it was initially associated with code level issues, TD can impact any type of software artifact and activity (Alves *et al*. 2016, Rios *et al*. 2018). For example, outdated requirement documentation can lead to a code that does not meet user requirements.

Despite the growing number of studies on TD, there is a clear concentration of studies investigating it from the

source code and its related artifacts perspective (Zazworka *et* al. 2014, Alves *et al*. 2016, Rios *et al*. 2018). Focusing solely on coding is risky business, because TD can affect many other software activities. But, how can one identify and manage TD related to different software activities?

This paper elaborates on the growing need to expand TD research to other areas of software development. It analyzes six elements related to TDM: causes, effects, preventive practices, reasons for non-prevention, repayment practices, and reasons for non-repayment of TD, for several types of software artifacts and activities. The paper uses a subset of the data collected by the InsighTD project, a family of surveys globally distributed on causes, effects, and management of TD (Rios *et al*. 2020). This data set consists of data from six countrywide replications of the survey, totaling 653 responses from software practitioners. By investigating how practitioners face TD in their projects, we gain insight into the state of practice regarding TDM, which allow us to identify existing gaps in TDM theory. The data are analyzed qualitatively and quantitatively to investigate whether the above listed TDM elements are more related to coding or to non-coding issues (e.g., planning and management, requirements engineering, human factors) of the software development.

This paper is based on our previous work by Berenguer *et al*. (2021), extending it by including:

- A more comprehensive analysis of the relation between TD and non-coding activities,
- Specializations of the hump diagram by process model (agile, hybrid, and traditional), and
- An analysis between TD, coding and non-coding activities by process model.

Our results indicate that both coding and non-coding activities are commonly affected by TD, but causes, effects, preventive practices, reasons for non-prevention, and reasons for non-repayment, affect non-coding activities more than coding activities. For repayment practices, we found similar behaviors between the two groups (coding and non-coding activities).

Given all the investigated TDM elements, some software development issues are more commonly reported by practitioners. *Planning and management issues* and *human factors* stand out, but there are several issues related to debt items such as *process*, *knowledge*, *TD management*, and *requirement engineering issues*.

Concerning the analysis per process models, we found that practitioners following agile, hybrid, or traditional process models shared a similar view on TD elements affecting coding activities. On the other hand, practitioners who use traditional process models have a different view of those using agile and hybrid process models on TD elements affecting non-coding activities. Results are presented with a hump diagram that, in combination with the analyses of each of the investigated TD management elements, provides guidance on what to expect from the presence of TD and how to react to them considering several issues of the software development process.

In addition to this introduction, this paper has seven additional sections. Section 2 presents background information on TD research and related work. Section 3 describes the methodology used. Section 4 presents the results of this work. And Section 5 presents the hump diagram and its specializations by process models. Section 6 summarizes the results and discusses their implications for researchers and practitioners. Section 7 discusses the threats to validity. Lastly, Section 8 presents our concluding remarks.

2 Background

TD can be incurred at any time and in several artifacts throughout the software development process. As such, it has different characteristics depending on the time it was incurred and the activities it is related to, such as testing, code, build, documentation, and so on (Alves *et al*. 2016). Although TD is a rising research topic, many studies focus solely on its relationship to source code.

Li *et al*. (2015) investigated studies on TD and its management (TDM), in addition to carrying out classification and thematic analysis on it, comprehensively understanding the concept of TD and presenting an overview of the current state of research in TDM. In their results, it was observed that code debt was the most cited type among the primary articles that were analyzed. In Alves *et al*. (2016), the authors also reported the focus on approaches to identify TD items from source code. The authors suggested that a possible explanation for this is that there is a plethora of tools that perform the analysis of source code that can be used to support the detection of TD.

In another study, Rios *et al*. (2018) presented fifteen types of TD. The authors also indicated that there is a concentration of studies focusing on source code. The authors gave some explanations for this phenomenon. The term TD was first coined by Cunningham (1992), who directly related it to source code, which may have influenced subsequent studies. Furthermore, the types related to code tend to cause effects that can be felt more quickly by development teams.

More recently, Saraiva *et al*. (2021) performed a systematic mapping study to investigate the current state of the art of TD tools, identifying which activities, functionalities and types of debt are handled by the existing tools to support TD management. The study identified 50 tools, 42 of which are new tools, and 8 tools extend an existing one. The main TD types addressed by tools deal with source code (60% - 30/50), architectural issues (40% - 20/50) and design issues (28% - 14/50). The distribution of tools over the categories was mainly: quantifying code properties, architectural smell detection, pattern matching, cost benefit analysis, project management, and code smell. The authors also reinforce that this trend is in line with the original definition of TD, which is heavily defined by concepts coming from source code and related issues.

Lenarduzzi *et al.* (2021) also performed a systematic mapping study to understand which TD prioritization approaches have been proposed in research and industry.

The results showed that code debt (38%), architecture debt (24%) and design debt (10%) are by far the most frequently investigated types of debt when considering TD prioritization, although there is scant evidence on other types like test and requirement debt. Thus, the approaches mainly involve models that reduce TD by acting on source code, removing or refactoring code smells or other metrics.

Such concentration of studies at the code level is a worrying scenario because other types of debt can also have impactful or even worse consequences on projects. We claim that it is necessary to go beyond the source code and investigate other facets of TD. We do it under the perspective of TD causes, effects, prevention, and repayment, and use data collected from InsighTD project, presented in the next section.

3 Research Method

This section presents the InsighTD project in which this work is contextualized, our research questions, and the data collection and analysis procedures.

3.1 The InsighTD project

InsighTD is a family of globally distributed industrial surveys, present in countries such as Brazil, Chile, Colombia, Costa Rica, the United States, and Serbia. It aims to investigate the causes, effects, and management of TD in software projects. Several results of the project have been disseminated so far, for example: the empirical design of the InsighTD and the results of its Brazilian replication on causes and effects of TD (Rios *et al*. 2020), probabilistic diagrams of causes and effects of TD (Rios *et al*. 2019), the set of causes and effects of TD collected from six InsighTD replications (Ramač *et al*. 2022, Freire *et al*. 2021b), the relation between TD and process models (Rios *et al*. 2021), TD prevention (Freire *et al*. 2020a, Freire *et al*. 2021a), and practices and impediments to repay TD items (Freire *et al*. 2020b, Perez *et al*. 2020, Freire *et al*. 2021a, Freire *et al*. 2021c). Other results from the project can be found at http://www.td-survey.com/publication-map/.

Concerning the relation between TD and development issues related to coding or other development issues, we previously investigated it in our previous work (Berenguer *et al*. 2021). In this paper, we further investigated it by including:

- A more comprehensive analysis of the relation between TD and non-coding activities, as shown in Section 4,
- Specializations of the hump diagram by process model (agile, hybrid, and traditional), as presented in Section 5, and
- An analysis between TD, coding, and non-coding activities by process model, as discussed in Subsection 5.2.

3.2 Research questions

In this work, we investigate whether TD management elements (causes, effects, prevention, and repayment) are more related to coding issues or to other software development issues. To this end, we consider the following research questions:

- **RQ1**: Are the causes of TD more related to coding issues or other software development issues?
- **RQ2**: Are the effects of TD more felt in coding issues or other issues in the software development process?
- **RQ3**: Is TD prevention more related to coding issues or other issues in the software development process?
- **RQ4**: Are the reasons for not preventing TD more related to coding issues or other development issues?
- **RQ5**: Is TD repayment more associated with coding issues or other issues in the software development process?
- **RQ6**: Are the reasons for not paying TD more related to coding issues or other development issues?

3.3 Data collection

This study uses a subset of available data from 18 questions from the InsighTD questionnaire. **Table 1** shows these questions, reports their type and the RQ they refer to.

Questions Q1 through Q8 document the characteristics of the survey respondents. More specifically, in Q8, the respondents inform the process model adopted in their projects, choosing one of the following options: **Agile** (a lightweight process that promotes iterative development, close collaboration between the development team and business side, constant communication, and tightly-knit teams); **Hybrid** (is the combination of agile methods with other non-agile techniques. For example, a detailed requirements effort, followed by sprints of incremental delivery); and **Traditional** (conventional document-driven software development methods that can be characterized as extensive planning, standardization of development stages, formalized communication, significant documentation and design up front). More information on the closed questions' options is available in Rios *et al*. (2020).

In Q13, respondents provide an example of a TD item that occurred in their projects. Participants discuss causes of TD in Q16 thru Q18 and effects in Q20. We use the answers given to these questions for answering RQ1 (Q16-Q18) and RQ2 (Q20). Concerning TD prevention, participants give their responses in Q22 and Q23, and address TD repayment in Q26 and Q27. The answers given in these questions are used for answering RQ3-4 (Q22 and Q23) and RQ5-6 (Q26 and Q27).

We invite only software practitioners from the Brazilian, Chilean, Colombian, Costa Rican, North American, and Serbian software industries through LinkedIn, industry-affiliated member groups, and industry partners for answering the survey.

Table 1. Subset of the InsighTD survey's questions (adapted from Rios *et al*. (2020)).

RQ	No.	Question (Q) Description	Type
-	**Q1**	What is the size of your company?	Closed
-	**Q2**	In which country are you currently working?	Closed
-	**Q3**	What is the size of the system being developed in that project? (LOC)	Closed
-	**Q4**	What is the total number of people of this project?	Closed
-	**Q5**	What is the age of this system up to now or to when your involvement ended?	Closed
-	**Q6**	To which project role are you assigned in this project?	Closed
-	**Q7**	How do you rate your experience in this role?	Closed
-	**Q8**	Which of the following most closely describes the development process model you follow on this project?	Closed
-	**Q10**	In your words, how would you define TD?	Open
-	**Q13**	Please give an example of TD that had a significant impact on the project that you have chosen to tell us about:	Open
RQ1	**Q16**	What was the immediate, or precipitating, cause of the example of TD you just described?	Open
RQ1	**Q17**	What other cause or factor contributed to the immediate cause you described above?	Open
RQ1	**Q18**	What other causes contributed either directly or indirectly to the occurrence of the TD example?	Open
RQ2	**Q20**	Considering the TD item you described in question 13, what were the impacts felt in the project?	Open
RQ3-4	**Q22**	Do you think it would be possible to prevent the type of debt you described in question 13?	Closed
RQ3-4	**Q23**	If yes, how? If not, why?	Open
RQ5-6	**Q26**	Has the debt item been repaid (eliminated) from the project?	Closed
RQ5-6	**Q27**	If yes, how? If not, why?	Open

3.4 Data analysis procedures

The analysis procedures are divided into three steps: demographics, preparing data for analysis, and data classification and analysis.

3.4.1 Demographics

We calculate the quantity of respondents choosing an option available through the closed questions of the survey. Subsequently, we sum up the participants' characterization.

3.4.2 Preparing data for analysis

For the open-ended questions, we applied coding process (Strauss and Corbin 1998). In answers given to Q16 thru Q18 and Q20, we used the coding process described in Rios *et al*. (2020) to identify a set of causes and effects, as well as the number of occurrences for each. To exemplify, let us consider the answers given by two respondents in Q16: "poorly developed code" and "low quality code". As these answers are associated with problems in source code, they were unified under the cause *sloppy code*.

We used the coding process described in Freire *et al*. (2020a) to code the responses to Q23. We identified practices for TD prevention from this process when Q22 received a positive response; otherwise, we identified reasons for TD non prevention. An example of this process is as follows: two respondents provided the following answers in Q23 when Q22 has a negative answer: "requirements are always going to change during development..." and "because when the client asks for features abruptly, no matter how generalized the architecture is towards the problem, with an outlier there may be, that can mean a refactor of the code, and that could dirty the code, reducing its maintainability". As these answers are associated with requirements change requests, they were unified under the reason for TD non-prevention *requirements change*.

Finally, we coded the responses to Q27 using the coding procedure described in Freire *et al*. (2020b). Similarly, if Q26 received a positive response, we identified TD repayment practices; otherwise, we identified non-repayment reasons. For both prevention and repayment, we also had a list of practices and reasons, and their corresponding number of occurrences. For example, two respondents provided the following answers in Q27 when Q26 has a positive answer: "we rewrote the offending code" and "it was fixed, code was refactored and greatly simplified". These answers were unified under the repayment practice *code refactoring*.

At least two researchers from each replication team participated in the coding process. The Brazilian replication team created the first codified list of causes, effects, prevention practices, reasons for not preventing, repayment practices, and reasons for not repaying, which was distributed to the other replication teams in order to standardize the used nomenclature. The consistency was verified by the Brazilian replication team.

3.4.3 Data classification and analysis

We began by analyzing the codes of each TD management element to determine whether they are related to coding issues or other software development issues. Repayment

practices such as *bug fixing*, *code refactoring*, and *code reuse*, for example, were classified as practices related to coding issues. However, the repayment practices *prioritizing TD items* and *updating system documentation* were linked to other software development issues. This procedure was carried out independently by the first and second authors. The third (prevention and repayment) and fourth (causes and effects) authors reached an agreement. The final classification was also reviewed by the last author.

Next, we classified the TD management elements related to the other software development issues using the grouping process defined by Strauss and Corbin (1998). The categories show the relationship between software development process issues (for example, requirement engineering issues, planning and management issues, and human factors issues) and each TD management element. The names of the categories are derived from the ongoing process of grouping the TD management elements around the central concern to which they are related. The causes *deadline* and *inappropriate planning*, for example, are part of the category planning and management issues, whereas the effects *team demotivation* and *dissatisfaction of the parties involved* are part of the category human factors. This procedure was carried out independently by the first and second authors. The third (prevention and repayment) and fourth (causes and effects) authors reached a consensus, and the final result was reviewed by the last author.

4 Results

Participants were asked to provide a definition of TD (Q10) and then an example of a significant TD item in their professional experience (Q13). As detailed in (Rios *et al.* 2020), the answers to the questions provided in Q13 were used as a criterion for the inclusion of participants. If they did not provide a valid example, their responses were discarded. In total, we considered the responses of 653 professionals from six countries (Brazil = 107, Chile = 89, Colombia = 134, Costa Rica = 145, Serbia = 79, US = 99). Next, we will present the characterization data of the participants, as well as the answers to the research questions posed in this study.

4.1 Demographics

Figure 1 presents the demographic information. Half of the participants identified themselves as developers, but managers (17%), testers (7%), software architects (13%), and other roles (13%) also answered the questionnaire. Besides, the participants described their experience level in their role. The majority of them is competent (good working and background knowledge of area of practice, with 34% of the total of participants), followed by proficient (depth of expertise of discipline and area of practice, 31%), expert (authoritative understanding of discipline and deep tacit information throughout area of practice, 21%), beginner (working information of key factors of practice, 12%), and novice (Minimal or "textbook" knowledge without connecting it to practice, 2%).

The majority of the participants worked in middle-sized companies (39%), followed by small (32%) and large (29%) ones. Further, participants normally worked in teams composed of 5-9 people (34%), but participants working in teams with 10-20 people (22%), less than five people (20%), more than 30 people (16%), and 21-30 people (8%) also answered the questionnaire. Concerning the process models adopted, the participants followed hybrid (45%), agile (42%), and traditional (13%) process models.

Regarding the systems, the respondents normally worked with systems with 10-100KLOC (35%), followed by ones with 100KLOC-1MLOC (30%), less than 10KLOC (14%), 1-10MLOC (14%), and more than 10MLOC (7%). Lastly, the majority of the systems is 2-5 years old, followed by 1-2 (23%) years old, less than one year old (17%), 5-10 years old (15%), and more than 10 years old (11%).

In summary, our data set is composed of answers from practitioners from different organization and team sizes,

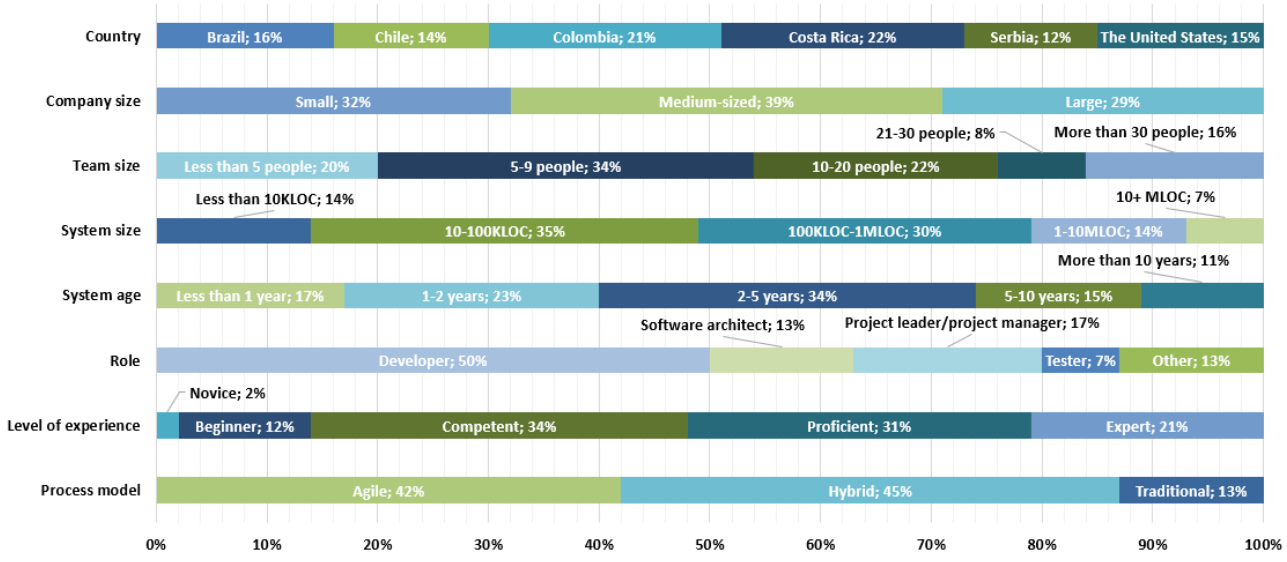

Figure 1. Participants' demographics.

system ages and sizes, roles, experience levels, and adopted process models.

In the following subsections, we present the detailed results of each investigated TD management element. We use the same structure when describing the results. For example, for the element TD cause, initially we (i) present the overall result. Next, we (ii) discuss the causes related to coding issues. Then, we (iii) present the causes related to the other software development issues, and (iv) analyze which are the types of those issues (e.g., planning and management, human factors, knowledge issues).

4.2 RQ1: Are the causes of TD more related to coding issues or other software development issues?

In total, 96 causes[1] that lead to the occurrence of TD were identified, totaling 1695 citations. Of this total, ~92% were related to other development issues, while only ~8% were related to code. This indicates a significant difference between the two subsets, representing a tendency of other software development issues to have an influence on the occurrence of TD items.

There are 13 causes related to coding. The ten most commonly cited are presented in the second column of **Table 2**. The complete list is available at https://bit.ly/37BopIF. The causes *non-adoption of good practices*, *sloppy code*, and *lack of refactoring* stand out. All of them indicate issues that compromise the internal quality of the product.

Alternatively, we identified 83 causes related to other software development issues. The three most commonly (third column of **Table 2**) cited reflect concerns focused on project management and planning: *deadline*, *not effective project management*, and *inappropriate planning*. Other issues related to the team's *lack of technical knowledge and experience*, *pressure*, and *processes* were also commonly mentioned.

We observed that those causes were related to each other and grouped them, identifying 14 categories of causes that reflect the main concerns that practitioners have during the development of software projects:

- Planning and management: refers to causes related to the project's planning and management issues. Some examples are *deadline*, *inappropriate planning*, and *not effective project management*;

Table 2. The 10 most cited causes related to coding and other software development issues.

	Coding		**Other development issues**	
	Cause	**#**	**Cause**	**#**
1st	Non-adoption of good practices	54	Deadline	169
2nd	Sloppy code	21	Not effective project management	98
3rd	Lack of refactoring	17	Inappropriate planning	83
4th	External component dependency	12	Lack of technical knowledge	80
5th	Adoption of contour solutions as definitive	11	Producing more at the expense of quality	67
6th	Lack of reuse practices	5	Inappropriate / poorly planned / poorly executed test	59
7th	Lack of automated testing	5	Lack of experience	58
8th	Discontinued component	4	Inaccurate time estimate	56
9th	Concern with just back-end development	4	Lack of qualified professional	54
10th	Inadequate data model	3	Pressure	53

- Human factors: groups causes related to people's participation in project issues. Some examples are *lack of experience* and *lack of commitment*;
- Knowledge issues: groups items originating from concerns around the knowledge of team members. Two examples are *lack of technical knowledge* and *lack of domain knowledge*;
- Requirements engineering: encompasses the causes related to requirements issues. Examples are: *change of requirements* and *requirements elicitation issues*;
- Verification, validation, and testing: encompasses the causes related to the execution of quality assurance activities. Two examples are *inappropriate/poorly planned/poorly executed test* and *lack of code review*;

[1] Some causes seem to overlap among them. For example, *non-adoption of good practices* could cover the causes *lack of refactoring* or *lack of reuse practices*. However, the cause *non-adoption of good practices* refers to the non-use of good practices that would facilitate the accomplishment and maintenance of activities in the project, as can be observed in the following responses from participants: “employment of bad design practices” and “lack of use of good software development practices”. On the other hand, *lack of refactoring* refers to situations in which the team does not perform the improvement of the internal structure of the code without changing its external behavior, as exemplified in “lack of code refactoring” and “there was no code refactoring at the beginning of the problem”. On its turn, *lack of reuse practices* occurs when existing software component or software component knowledge is not used for the construction of a new software, for example, “need to create the culture of reusability”. Another example of overlapping encompasses the causes *not effective project management* and *inappropriate planning*. However, the cause *not effective project management* refers to inadequate management during project development, as reported in: “not following planning” and “lack of understanding of managers”. Differently, the cause *inappropriate planning* refers to issues in project planning, for example, “lack of prioritization of activities” and “deficiency in project planning (disorganization)”.

- Architectural issues: groups causes related to decisions made regarding software architecture. Examples are: *inadequate technical decisions* and *problems in architecture*;
- Process issues: refers to causes related to the definition or execution of the processes used in the development of the software. Two examples are *lack of a well-defined process* and *lack of traceability of bugs*;
- Design issues: encompasses causes related to the design of the software. There are two causes in this category: *poor design* and *changes in design*;
- Documentation: groups causes related to documentation. Example of causes in this category are *nonexistent documentation* and *outdated/incomplete documentation*;
- External factors: refers to causes associated with external factors, such as *customer does not listen the project team* and *structural change in the involved organizations*;
- Infrastructure issues: encompasses causes related to problems in the software development infrastructure, such as *required infrastructure unavailable* and *updating existing tools*;
- Organizational issues: groups causes from the organizational context, such as *lack of awareness of the importance of testing and refactoring* and *organizational misalignment*;
- Quality issues: refers to causes (*lack of quality*) associated with lack of quality in software artifacts;
- TD Management: encompasses causes related to management of TD items. This category has only the cause *lack of perception of the importance of dealing with TD*.

Table 3 shows the categories together with the corresponding number of causes, number of citations, and percentage of the causes cited in relation to the other categories. The category planning and management stood out with ~47% of citations, representing more than three times the citations of the second ranked category. This is an indication that the causes of the occurrence of TD are strongly related to project management issues. The results also highlight the importance that human factors have, occupying the second position with ~13% of citations. This result is somehow aligned with previous work on social debt (Tamburri *et al*. 2015, Martini *et al*. 2019). Concerns related to requirements engineering and issues related to knowledge were also commonly mentioned.

4.3 RQ2: Are the effects of TD more felt in coding issues or other issues in the software development process?

The participants reported a total of 73 TD effects, totaling 980 citations. Among them, ~64% are related to other development issues and ~36% are related to coding.

Table 3. Categories of causes related to other software development issues.

Categories of causes	#causes	#cited causes	~%cited causes
Planning and Management	22	733	47%
Human Factors	10	206	13%
Knowledge Issues	7	128	9%
Requirement Engineering	7	120	8%
VV&T	6	91	6%
Architectural Issues	6	63	5%
Process Issues	6	54	4%
Design Issues	2	45	3%
Documentation	4	37	2%
External Factors	4	25	2%
Organizational Issues	3	25	2%
Infrastructure Issues	4	15	1%
Quality Issues	1	12	1%
TD Management	1	1	0.1%

There are 18 coding-related effects experienced by the participants. The 10 most commonly cited are presented in **Table 4** (second column). The full list is available at https://bit.ly/37BopIF. Concerns about the capacity of the team to evolve the code, rework, and the need of employing refactoring practices to improve the internal quality of the software are common. Other common effects are: *bad code*, *low performance*, and *stop development for debt repayment*.

Table 4. The 10 most cited effects related to coding and other development issues.

	Coding		Other development issues	
	Effects	#	Effects	#
1st	Low maintainability	97	Delivery delay	141
2nd	Rework	86	Low external quality	78
3rd	Need of refactoring	35	Financial loss	55
4th	Bad code	31	Increased effort	41
5th	Low performance	28	Stakeholder dissatisfaction	34
6th	Stop dev. activities for debt repayment	14	Team demotivation	24
7th	Increase in the amount of maint. activities	13	Stress with stakeholders	23
8th	Difficulty in impl. the system	10	Team overload	16
9th	Low code reuse	8	Fall in productivity	13
10th	Low reliability	7	Project not completed	13

We identified 55 effects related to other development issues. The four most commonly (third column of **Table 5**) cited reflect concerns on the project management and planning (*delivery delay*, *increased effort*, *financial loss*) and external quality of the product (*low external quality*). Issues related to human factors were also commonly cited, with emphasis on *stakeholder dissatisfaction*, *team demotivation*, and *stress with stakeholders*.

Table 5 shows the categories of effects related to other software development issues. The category planning and management has ~47% of citations, revealing that managerial aspects of software development are commonly affected by the presence of debt items. Next is the human factor category, with ~18% of the effects cited, showing that TD also impacts human aspects of software development. Quality issues are also a common concern. The other categories are less commonly cited.

Table 5. Categories of effects related to other software development issues.

Categories of effects	#effects	#cited effects	~%cited effects
Planning and Management	15	297	47%
Human Factors	7	110	18%
Quality issues	6	110	18%
VV&T	3	23	4%
Design Issues	2	21	3%
Knowledge issues	8	21	3%
Architectural Issues	4	18	3%
Organizational issues	3	10	2%
Documentation	1	6	1%
Process Issues	2	4	1%
Requirement Engineering	2	4	1%
Infrastructure Issues	1	3	0.5%
TD Management	1	2	0.3%

4.4 RQ3: Is TD prevention more related to coding issues or other issues in the software development process?

The data shows a total of 89 practices to support the prevention of TD items, resulting in 819 citations. From this, ~84% are items related to other development issues, while only ~16% are associated with code. This result indicates a tendency for other development issues to play a key role in the prevention of TD.

We identified a total of 13 TD prevention practices related to coding. **Table 6**, second column, presents the 10 most cited items. The complete list is available at https://bit.ly/37BopIF. *Adoption of good practices*, *using good design practices*, *refactoring*, *code review*, *increasing time for analysis and design*, *use the most appropriate version of the technology*, and *appropriate reusing of code* are the prevention practices most cited by the participants. The *adoption of good practices* and *using good design practices* reflect concerns that practitioners should have when carrying out their coding and design activities. The practices *refactoring* and *code review* are related to the continuous improvement of the code under development. Lastly, *increasing time for analysis and design*, *use of the most appropriate version of the technology*, and *appropriate reusing of code* are related to concerns that teams must have around an adequate analysis of the functionalities, implementation of the software structure, and software reuse, respectively.

Table 6. Top 10 most commonly cited TD prevention practices related to coding or other development issues.

	Coding		Other development issues	
	Prevention Practices	**#**	**Prevention Practices**	**#**
1st	Adoption of good practices	49	Well-defined requirements	57
2nd	Using good design practices	26	Better Project Management	43
3rd	Refactoring	12	Providing training	36
4th	Code review	10	Follow the proj. planning	34
5th	Increasing time for analysis and design	7	Improving software development process	33
6th	Use the appropriate version of the tech.	7	Improve documentation	26
7th	Appropriate reusing of code	6	Well planned deadlines	26
8th	Version control	5	Better project planning	24
9th	Considering technical constraints	4	Creating tests	24
10th	Improving the project maintainability	4	Allocation of qualified professionals	23

On the other hand, we found 76 prevention practices related to other development issues. **Table 6** (third column) shows the ten most cited. Interestingly, five of them reflect different concerns through the software development process, such as management (*following the project planning* and *better project management*), the process itself (*improving software development process*), the documentation (*well-defined requirements*), and the qualification of the team (*providing training*). We see in **Table 7** that TD prevention practices are commonly related to project management issues (~34%). The results also highlight the importance that the process followed by the team has, ranking second (~12%) among the most cited categories. Concerns related to requirements, VV&T, TD management, and human factors were also commonly mentioned.

Table 7. Categories of prevention practices related to other software development issues.

Categories of prevention practices	#practices	#cited practices	~%cited practices
Planning and Management	21	232	34%
Process Issues	8	80	12%
Requirement Engineering	5	69	11%
VV&T	11	67	10%
TD Management	7	64	10%
Human Factors	11	61	9%
Knowledge Issues	4	51	8%
Documentation Issues	2	28	4%
Architectural Issues	3	27	4%
Organizational Issues	2	4	1%
Infrastructure Issues	2	3	1%

4.5 RQ4: Are the reasons for not preventing TD more related to coding issues or other development issues?

Participants reported 25 reasons that lead to the non-prevention of TD items, resulting in 63 citations. Of them, ~87% are related to other development issues, while only eight ~13% are related to coding. Again, other development issues have an important role in preventing TD.

There are only four reasons related to code leading teams not to prevent the occurrence of debt items: *lack of technical knowledge*, *lack of good technical solutions*, *lack of concern about maintainability*, and *continuous change of coding standards*. On the other hand, we found 21 reasons (the 10 most cited are presented in **Table 8**) related to other software development issues. *Short deadline* was the most cited.

Table 8. Top 10 most cited reasons for not preventing TD related to other development issues.

Other Development Issues					
	Reason	**#**		**Reason**	**#**
1st	Short deadline	14	6th	Documentation issues	2
2nd	Ineffective management	7	7th	Lack of process maturity	2
3rd	Lack of predictability in the soft. development	5	8th	Lack of qualified professionals	2
4th	Requirements change	5	9th	Legacy system difficult to heal	2
5th	Pressure for results	4	10th	Accepting the TD	1

Table 9 shows the categories identified. Planning and management once again stands out with ~38% of citations. The other categories were less commonly cited, with less than seven citations. Although not too mentioned, the result suggests that other issues related to the software development can also negatively influence teams in TD prevention.

Table 9. Categories of reasons for TD non-prevention related to other software development issues.

Categories of reasons	#reason	#cited reasons	~%cited reasons
Planning and Management	2	21	38%
Requirement Engineering	2	6	11%
Coding	1	5	9%
External Factors	2	5	9%
Human Factors	4	4	8%
Process Issues	2	3	6%
Design Issues	1	2	4%
Documentation Issues	1	2	4%
Knowledge Issues	1	2	4%
TD Management	2	2	4%
Architectural Issues	1	1	2%
Infrastructure Issues	1	1	2%
Organizational Issues	1	1	2%

4.6 RQ5: Is TD repayment more associated with coding issues or other issues in the software development process?

We identified 32 TD repayment practices, resulting in 315 citations. Of them, ~56% are related to other development issues, while ~44% are associated with code. Unlike the other TD management elements, these percentages differ slightly, indicating that coding issues play a key role in TD repayment initiatives.

We recognized eight TD repayment practices related to coding, presented in **Table 10**. *Code refactoring* and *design refactoring* are the most cited practices. Both are associated with changes in the internal structure of the system without changing its external behavior. The practices *solving technical issues* and *bug fixing* focus on fixing open issues in the code. Lastly, the practices *using code analysis*, *code reviewing*, and *using code reuse* can support teams implementing TD repayment initiatives, i.e., although these practices did not repay the debt, they increase the capacity for better repayment.

The remaining 24 repayment practices are related to other development issues. **Table 10** (third column) shows the ten most cited ones. These practices evidence several concerns in software development processes: documentation (*update system documentation*), organizational decisions (*hiring*

specialized professionals), project management (*increasing the project budget*, *monitoring and controlling project activities*, *negotiating deadline extension*, *investing effort on TD repayment*, and *prioritizing TD items*), process (*improving the development process* and *using short feedback iterations*), and software quality (*investing effort in testing activities*).

Table 10. Top 10 most commonly cited TD repayment practices related to coding or other development issues.

	Coding		**Other Development Issues**	
	Repayment practices	**#**	**Repayment practices**	**#**
1st	Code refactoring	80	Investing effort on TD repayment activities	33
2nd	Design refactoring	25	Investing effort on testing activities	22
3rd	Adoption of good practices	10	Prioritizing TD items	15
4th	Solving technical issues	9	Negotiating deadline extension	14
5th	Bug fixing	6	Update system documentation	9
6th	Using code analysis	3	Monitoring and controlling project activities	9
7th	Code reviewing	3	Increase the project budget	9
8th	Using code reuse	2	Improving the development process	8
9th	-		Hiring specialized professionals	8
10th	-		Using short feedback iterations	7

Table 11 presents the categories of repayment practices. TD management and planning and management stand out with ~32% and ~27% of the total of citations. The categories verification, validation and test, and process issues were both cited by ~12% of participants, while the others were less commonly reported.

4.7 RQ6: Are the reasons for not paying TD more related to coding issues or other development issues?

We identified 27 reasons for not repaying TD items, totaling 319 citations. From these, 99.7% are related to other development issues and only *lack of access to the component code* (0.3%) is associated with code. The reasons for TD non-repayment arise from development issues other than coding.

Table 11. Categories of repayment practices related to other software development issues.

Categories of repayment practices	**#practices**	**#cited practices**	**~%cited practices**
TD Management	4	56	32%
Planning and Management	8	47	27%
VV&T	1	22	13%
Process Issues	5	21	12%
Documentation	1	9	6%
Organizational issues	1	8	5%
Human Factors	1	6	4%
Requirement Engineering	1	3	2%
Infrastructure Issues	1	3	2%
Design Issues	1	2	1%

Table 12 shows the ten best-positioned reasons for not repaying TD. The complete list is available at https://bit.ly/37BopIF. We notice that the majority of the reasons (*focusing on short-term goals*, *lack of time*, *cost*, *lack of resources*, *effort*, *the project was discontinued*, *complexity of the TD item*, and *insufficient management view about TD repayment*) are associated with project planning and management. The others refer to external (*customer decision*) and human (*team overload*) factors.

Table 12. Top 10 most cited reasons for not paying off TD related to other development issues.

Other Development Issues					
	Reason	**#**		**Reason**	**#**
1st	Focusing on short term goals	69	6th	Customer decision	13
2nd	Lack of org. interest	48	7th	Complexity of the TD item	12
3rd	Lack of time	41	8th	Effort	11
4th	Cost	34	9th	Insufficient mgmt. view on TD repayment	10
5th	Lack of resources	19	10th	Complexity of the project	10

The reasons were also grouped into categories. Planning and management issues stand out with ~58% of citations, as shown in **Table 13**, pointing out that the reasons for this category are categorical for TD non-repayment. The categories *organizational issues* and *TD management* were also commonly cited by ~16% and ~11% of the participants.

Table 13. Categories of reasons for TD non-repayment related to other software development issues.

Categories of reasons	#reason	#cited reasons	~%cited reasons
Planning and Management	7	185	58%
Organizational issues	2	50	16%
TD Management	7	34	11%
External Factor	1	13	5%
Knowledge issues	3	12	4%
Human Factors	3	11	4%
Architectural Issues	2	11	4%
VV&T	1	2	1%

5 Organizing the TD Management Elements into Hump Diagrams

We represent the relationship between the investigated TD management elements (causes, effects, prevention practices, reasons for TD non-prevention, repayment practices, and reasons for TD non-repayment) and software development issues in hump diagrams (**Figure 2**).

To plot results for coding and for other issues in the same hump diagram, we normalized the number of citations for an element of a specific software development issue with the total number of citations for that element. For example, prevention practices have in total 819 citations, but 232 citations for the issue *planning and management*. Thus, the hump value for *planning and management issues* of prevention practices is 28% (232/819*100). This count is slightly different from the ones we used in Tables 3, 5, 7, 9, 11, and 13 because now we consider *coding* as another software development issue.

5.1 Using the diagram

We can read the hump diagram horizontally and vertically. Horizontally, we have a broad view on the impact of each software development issue through the TD management elements. For example, in **Figure 2**, we can notice that *coding* plays an important role for all the analyzed TD elements, but mainly for TD repayment. There is a high concentration of practices related to TD repayment and, at the same time, almost none of reasons for the non-repayment of debt items is due to coding issues.

We also perceive that there are many other issues we need to be aware of when dealing with TD in software projects, mainly, *planning and management*. Indeed, this is even stronger when combined with *TD management* concerns. Much about the non-repayment of TD can be understood by looking at these issues.

Human factors also call our attention, clearly indicating that TD, more than technical aspects of the software development, is also about team morale, satisfaction, motivation, communication, and commitment. Other commonly found issues in several elements of the TD management are *architectural issues*, *design issues*, *documentation*, *knowledge issues*, *process issues*, *requirement engineering*, and *VV&T*.

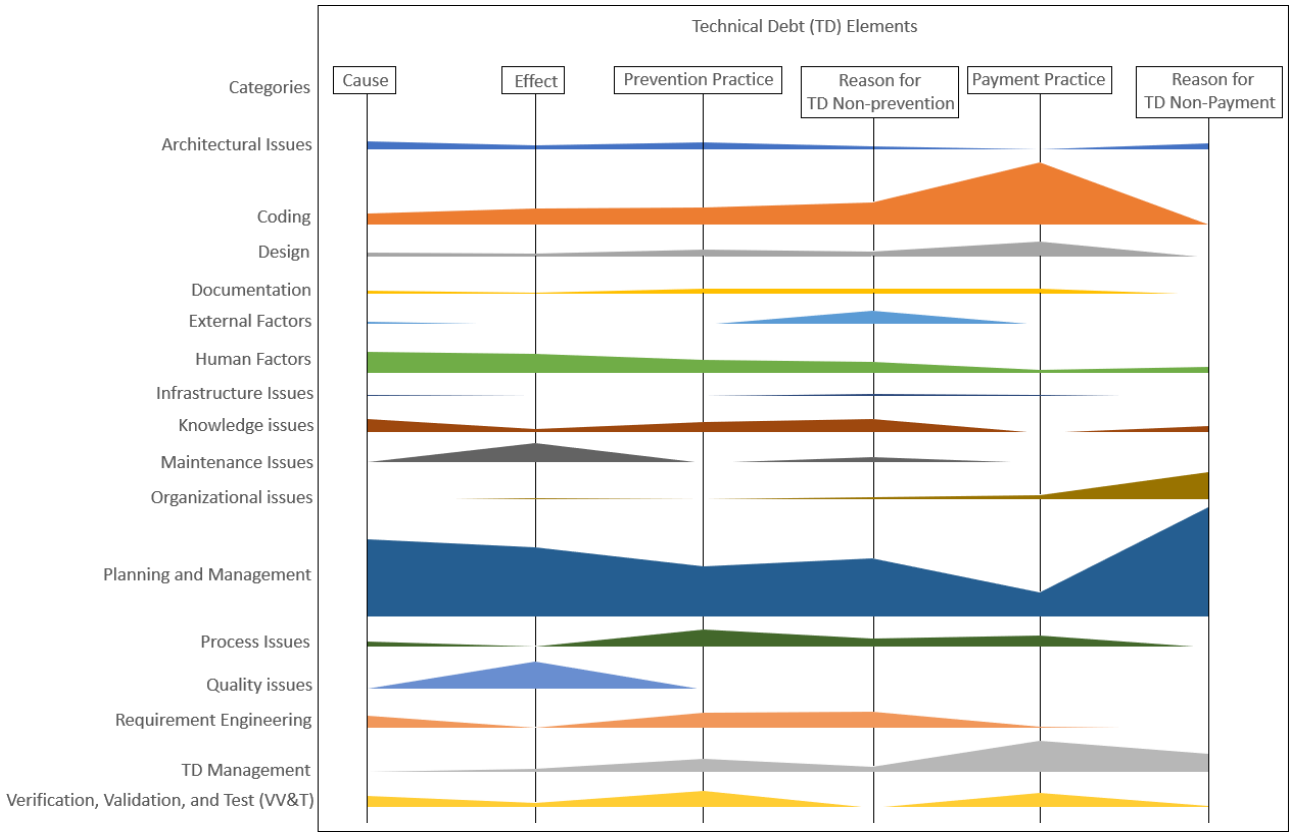

Figure 2. The hump diagram for TD management elements and software development issues.

By reading the diagram vertically, we can observe the impact of all identified software development issues on each TD management element. In **Figure 2**, for example, we can observe that *planning and management*, *organizational*, and *TD management issues* are decisive for the non-repayment of debt items. We also notice that the presence of debt items mainly impacts (effect) *planning and management*, *quality issues*, *maintenance issues*, *human factors*, and *coding*.

Practitioners can use the hump diagram to have a comprehensive view on how TD relates to several issues of their software projects, ranging from organizational to coding level issues. Moreover, for each TD management element, they can go through the detailed results presented in Section 4 and the auxiliary material to understand how to deal with them. For example, by looking at **Figure 2**, a practitioner can see that the effects of TD are commonly related to *coding*, *human factors*, *maintenance*, *quality*, and *planning and management* issues. If (s)he is interested in discovering more about the *human factors* issues, then (s)he can observe in the results and auxiliary material that *team demotivation*, *dissatisfaction of the parties involved*, and *stress with stakeholders* are the main concerns to be mitigated.

5.2 Specializing the diagram by process models

Practitioners can specialize the hump diagram for their context. To illustrate it, we organize the TD management elements considering the process model used by the participants who answered the InsighTD questionnaire choosing one of the following options: agile, hybrid, and traditional.

Figures 3, **4**, and **5** present the hump diagram for agile, hybrid, and traditional process models, respectively. Comparing them, we can notice that the diagrams for agile and hybrid process models are just slightly different from each other. It indicates that the view on the TD management elements goes in the same direction to these process models. Conversely, traditional process model presents some particularities against the other models. For example, prevention practices are more affected by architectural, infrastructure, organizational, and requirement engineering issues in traditional process model than the others. Reasons for TD non-prevention are less affected by coding, design, documentation, human factors, knowledge, maintenance, requirement engineering, and TD management in traditional process model, while external factors and planning and management affect mainly this model.

To further understand the possible impact of different process models in the TD management elements, we organized ranked lists of each TD management element considering its number of citations by process models (agile, hybrid, and traditional). To verify if there are differences between the lists, we adopted the RBO (rank-biased overlap) analysis (Webber *et al*. 2010), which quantitatively measures how similar the ranked lists are.

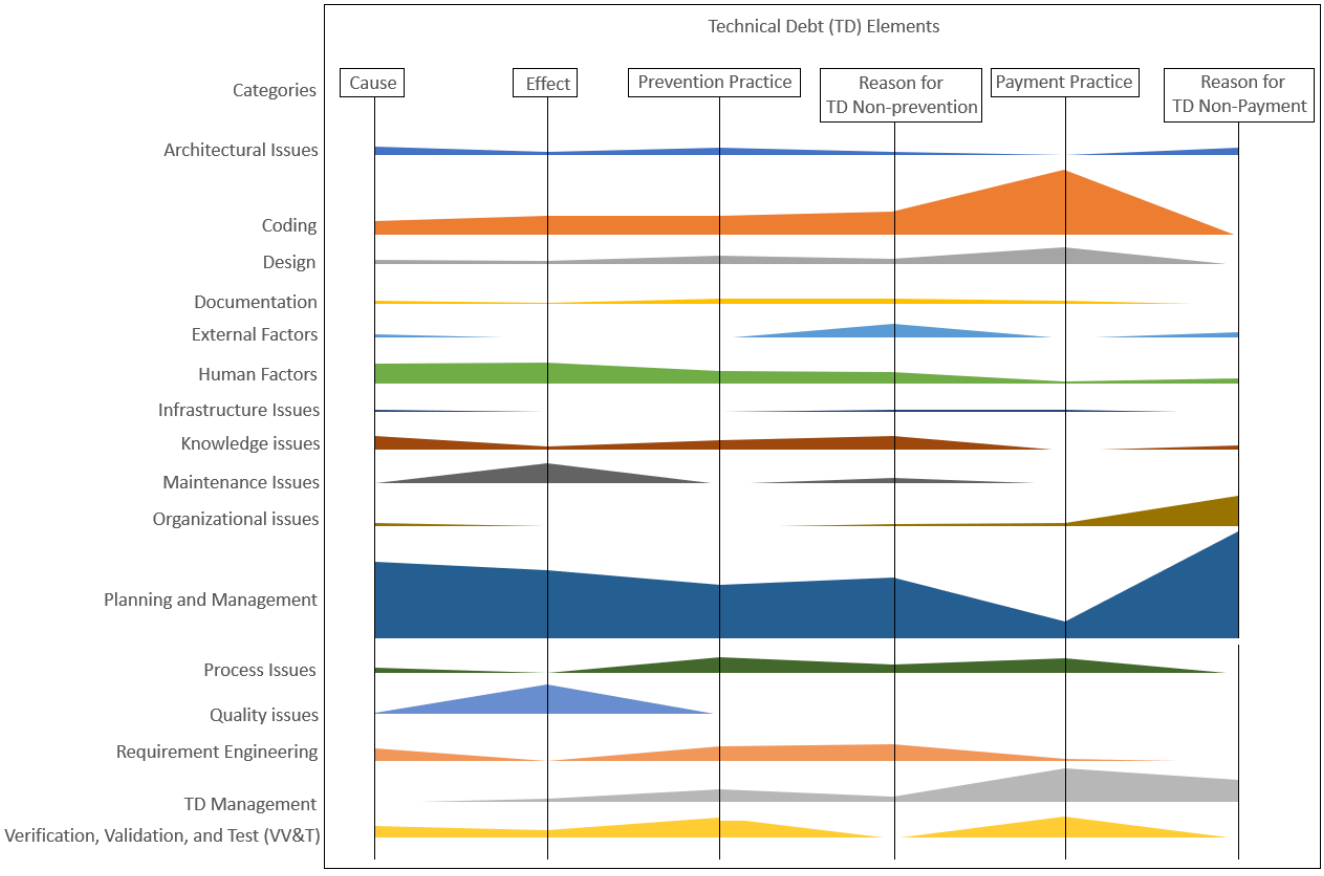

Figure 3. The hump diagram for agile model process.

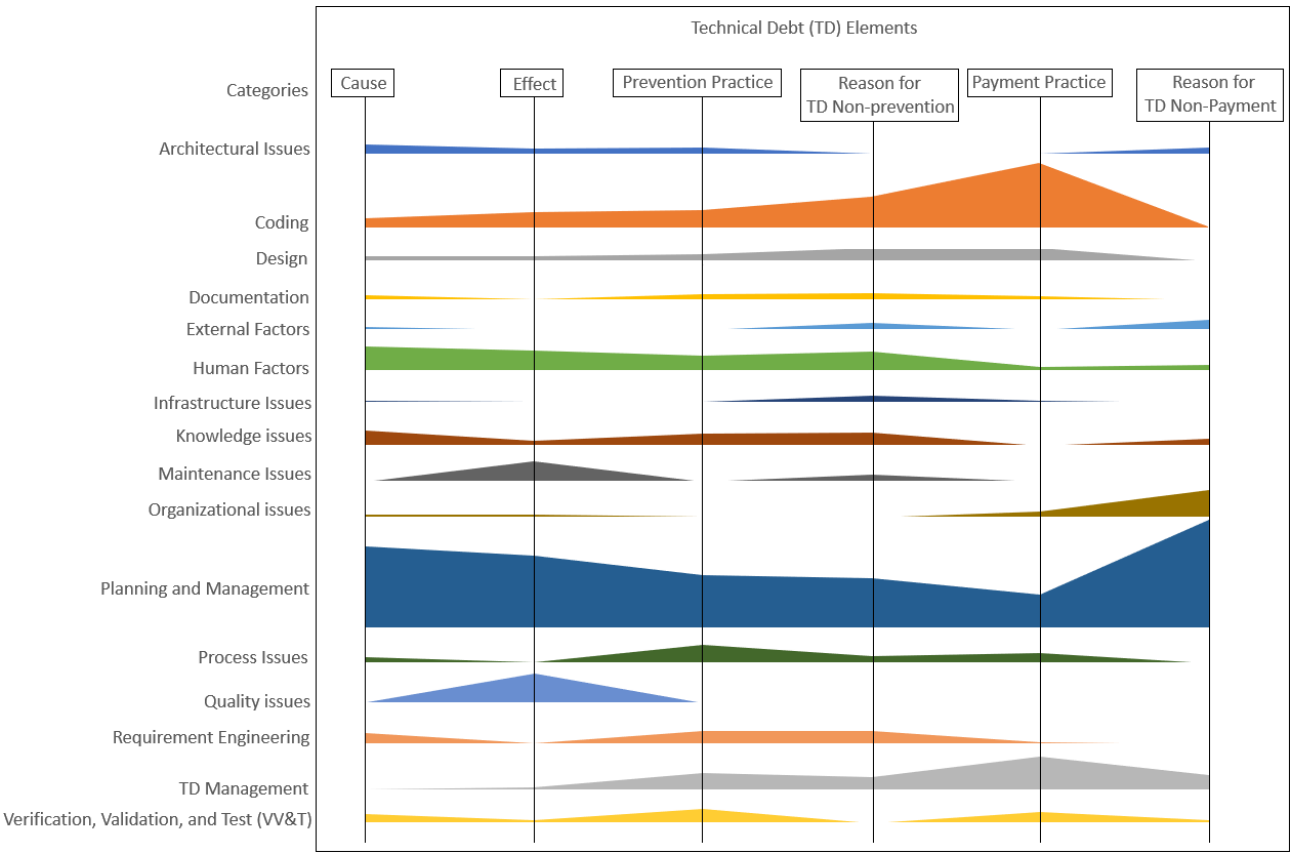

Figure 4. The hump diagram for hybrid model process.

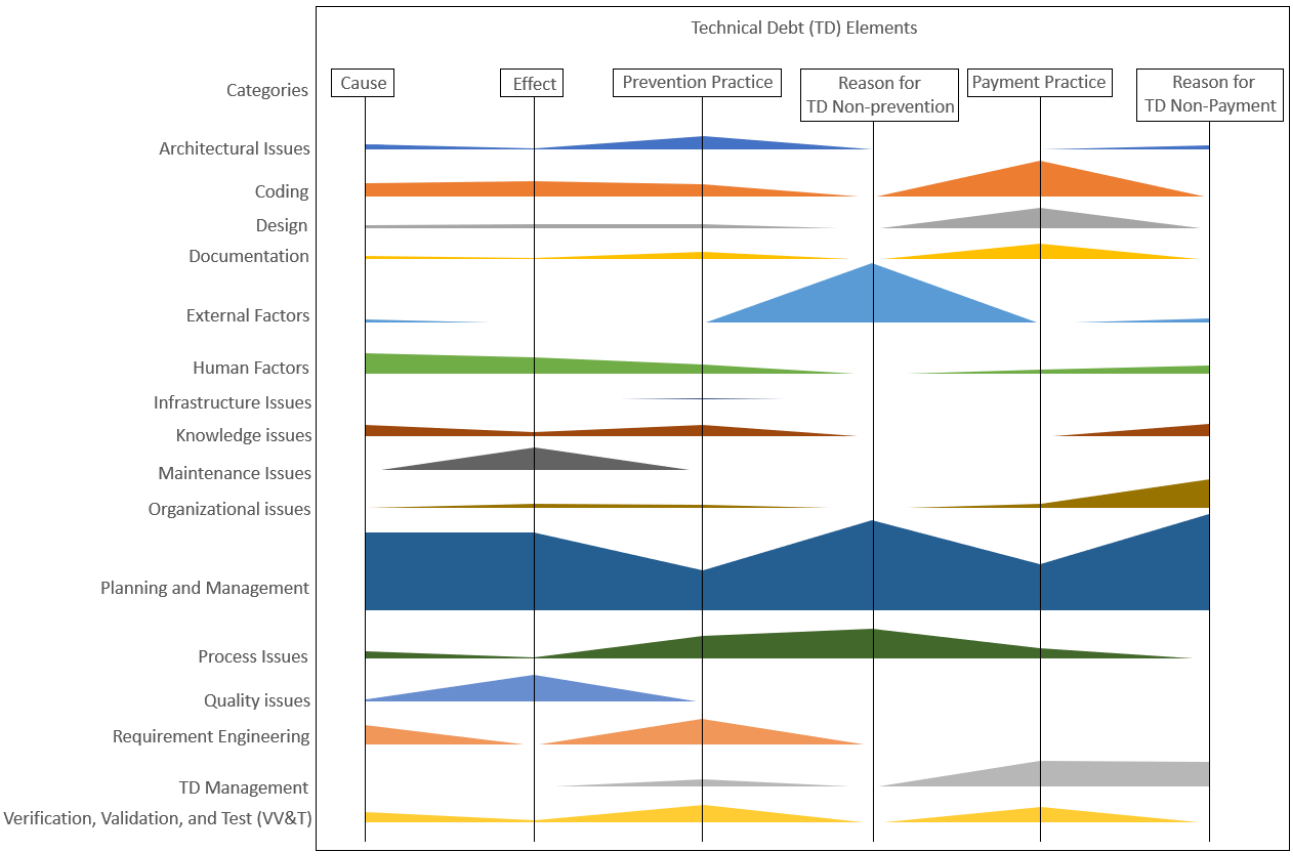

Figure 5. The hump diagram for traditional model process.

RBO gives a value ranging from 0 to 1. The closer this value is to 1, the greater the similarity between the lists. As RBO supports top-weighted ranked lists, the first elements of a list have more impact on the similarity index than the last ones. We can configure what elements will be compared by setting the *p-value*, which, differently than the p statistic, refers to a level of overlapping and the degree of top-weightedness. In the analysis, we chose *p-value* ranging from 0.5 (only the very initial elements of a rank are considered) to 0.9 (almost all elements are considered). The results of the comparison for each of the TD management elements are presented in the following subsections.

5.2.1 Comparing TD causes between agile, hybrid, and traditional process models

Figure 6 shows the results of the comparison between the ranked lists of causes for each process model considering (A) causes related to coding issues and (B) causes related to other software development issues. The RBO analysis for causes related to coding (**Figure 6 (A)**) reveals that the similarity level is about 80-90% between the three lists. It indicates that the lists are quite similar with little variation when more causes are included, i.e., the *p-value* increases.

This similarity can be perceived when we observe the top-5 ranked causes for each process model (**Table 14**). The cause *non-adoption of good practices* was the most cited cause for all process models, while *lack of refactoring*, *sloppy code*, *adoption of contour solutions as definitive* were perceived, but in different positions. For example, *lack of refactoring* (agile: 2nd, hybrid: 4th, and traditional: 3rd) and *sloppy code* (agile: 3rd, hybrid and traditional: 2nd). Further, we can see that the cause *external component dependency* is not perceived in traditional process model while *lack of reuse practices* is only perceived in this process model.

For causes related to other software development issues (**Figure 6 (B)**), we can see that the RBO value is almost constant with similarity level about 80-90% for agile and hybrid process models. Differently, the similarity level is about 65-80% when comparing traditional with agile/hybrid. In **Table 15**, we can see that the cause *deadline* was the most cited cause for each process model. Regarding agile and hybrid process models, they did not share the causes *focus on producing more at the expense of quality* and *lack of experience*. However, the causes *inaccurate time estimate*, *inappropriate planning*, and *lack of qualified professional* were perceived only in the context of traditional process model.

Table 14. Top 5 most cited causes related to coding issues per process model.

	Agile	Hybrid	Traditional
1	Non-adoption of good practices (25)	Non-adoption of good practices (23)	Non-adoption of good practices (6)
2	Lack of refactoring (10)	Sloppy code (8)	Sloppy code (5)
3	Sloppy code (8)	**External component dependency (7)**	Lack of refactoring (2)
4	Adoption of contour solutions as definitive (6)	Lack of refactoring (5)	**Lack of reuse practices (2)**
5	**External component dependency (4)**	Adoption of contour solutions as definitive (4)	Adoption of contour solutions as definitive (1)

Table 15. Top 5 most cited causes related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Deadline (66)	Deadline (85)	Deadline (18)
2	Inappropriate planning (35)	Not effective project management (53)	**Inaccurate time estimate (14)**
3	Not effective project management (35)	Inappropriate planning (38)	Inappropriate / poorly planned / poorly executed test (13)
4	Lack of technical knowledge (34)	Lack of technical knowledge (38)	**Inappropriate planning (10)**
5	**Focus on producing more at the expense of quality (30)**	**Lack of experience (32)**	**Lack of qualified professional (10)**

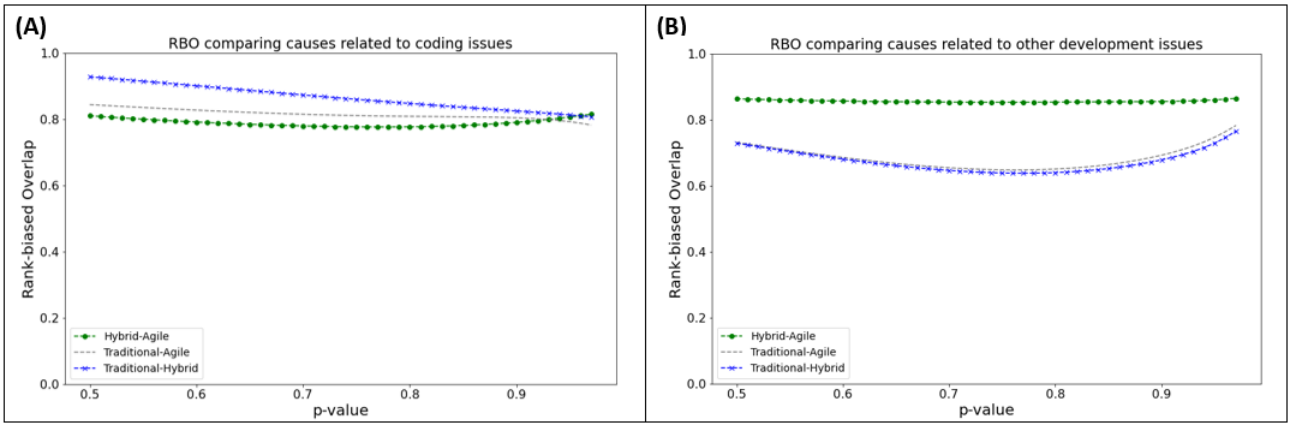

Figure 6. RBO comparing causes related to (A) coding and (B) other software development issues.

In summary, coding-related causes are perceived in the same way in agile, hybrid, and traditional process models, while non-coding related causes are differently perceived by those who follow traditional process models.

5.2.2 Comparing TD effects between agile, hybrid, and traditional process models

Figure 7 shows the results of the comparison between the ranked lists of effects by process model considering (A) coding related effects and (B) effects related to other software development issues. The RBO analysis for effects related to coding (**Figure 7 (A)**) reveals that the lists are quite similar, as the similarity level is about 90% between the three lists.

Analyzing the top 5 ranked effects of each process model (**Table 16**), we can see this similarity. For example, the effects *low maintainability* and *rework* were the most cited effects for all process models, occupying the same position in the lists. Further, the effect *difficulty in implementing the system* is only perceived by the traditional process model while it did not perceive the effect *need for refactoring*.

Table 16. Top 5 most cited effects related to coding issues per process model.

	Agile	Hybrid	Traditional
1	Low maintainability (40)	Low maintainability (43)	Low maintainability (14)
2	Rework (39)	Rework (35)	Rework (12)
3	**Need for refactoring (19)**	Bad code (17)	Bad code (5)
4	Low performance (14)	**Need for refactoring (14)**	Low performance (4)
5	Bad code (9)	Low performance (10)	**Difficulty in implementing the system (3)**

Regarding the effects related to other software development issues (**Figure 7 (B)**), the similarity level is almost 100% for the first effects in the agile and hybrid lists. It means that these process models have the same view on the most critical effects of TD, but this similarity level decreases when more effects are considered.

Table 17 presents the top 5 ranked effects by process models. We can see that the effect *delivery delay* was the most perceived effect by the process models. Besides, the effects from the list of agile and hybrid process models are quite the same, except *team demotivation* and *stakeholder dissatisfaction*. Although the effect *design problems* is only perceived in the context of traditional process models, the other effects (*financial loss*, *low external quality*, and *team demotivation*) are also present in the other two lists.

In conclusion, agile, hybrid, and traditional process models are related to almost the same coding-related effects. This also applies for non-coding related effects.

Table 17. Top 5 most cited effects related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Delivery delay (51)	Delivery delay (69)	Delivery delay (21)
2	Low external quality (34)	Low external quality (36)	Financial loss (10)
3	Financial loss (20)	Financial loss (25)	Low external quality (8)
4	Increased effort (18)	Increased effort (20)	Team demotivation (5)
5	**Team demotivation (13)**	**Stakeholder dissatisfaction (19)**	**Design problems (3)**

5.2.3 Comparing TD preventive practices between agile, hybrid, and traditional process models

Figure 8 shows the results of the comparison between the ranked lists of preventive practices by process model considering (A) preventive practices related to coding and (B) those related to other software development issues. The RBO analysis for preventive practices related to coding (**Figure 8 (A)**) reveals that the lists are different. The similarity level is about 60-80% between the three lists.

In **Table 18**, we can see that while the preventive practice *adoption of good practices* was the most used practice in the process models, the other practices were not shared by all

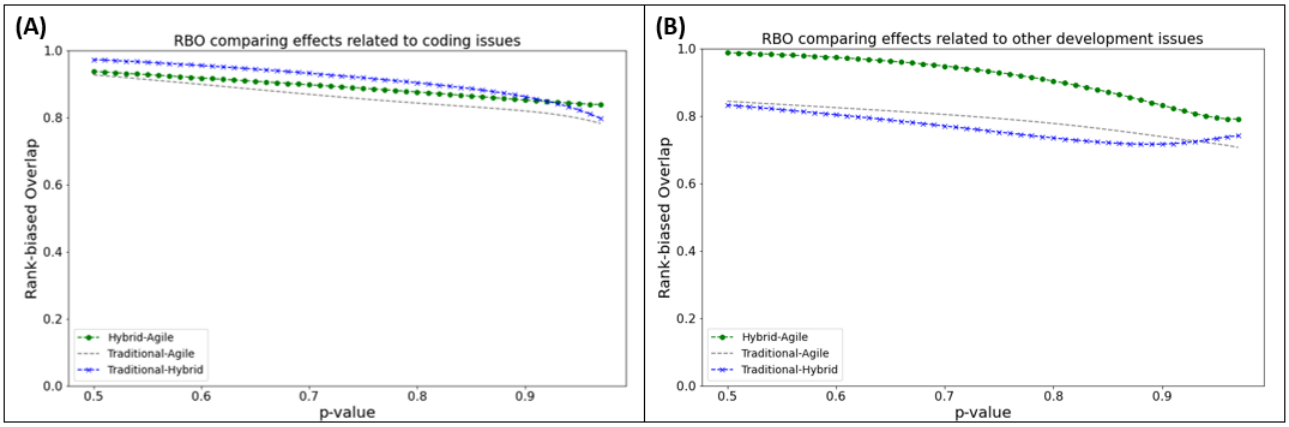

Figure 7. RBO comparing effects related to (A) coding and (B) other software development issues.

process models. For example, *using good design practices*, *refactoring* and *considering technical constraints* are only present in the context of agile process model, while *use the most appropriate version of the technology* and *bug tracking* are only related to traditional process model.

Table 18. Top 5 preventive practices related to coding issues per process model.

	Agile	Hybrid	Traditional
1	Adoption of good practices (18)	Adoption of good practices (25)	Adoption of good practices (6)
2	**Using good design practices (13)**	Appropriate reusing of code (3)	Increase time for analysis and design (2)
3	**Refactoring (8)**	Code review (2)	**Use the most appropriate version of the technology (2)**
4	Code review (7)	**Improving the maintainability of the project (4)**	Appropriate reusing of code (1)
5	**Considering technical constraints (4)**	Increase time for analysis and design (3)	**Bug tracking (1)**

Concerning the preventive practices related to other software development issues, the similarity level is 70-80% (**Figure 8 (B)**), indicating that the lists are also different. In **Table 19**, we can see that the preventive practice *well-defined requirement* was present in all process models, but the others were not shared by all process models. For instance, *well-defined architecture*, *creating tests*, and *improve documentation* were only used by traditional process models.

In summary, agile, hybrid, and traditional process models did not share the same view on preventive practices regardless they are related to coding or not.

Table 19. Top 5 most cited preventive practices related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Well-defined requirement (21)	Well-defined requirement (26)	Well-defined requirement (10)
2	**Following the project planning (17)**	Better Project Management (22)	**Well-defined architecture (6)**
3	Better Project Management (16)	Training (18)	Better Project Management (5)
4	Training (13)	**Improving software development process (17)**	**Creating tests (5)**
5	**Better project planning (12)**	**Well planned deadlines (14)**	**Improve documentation (5)**

5.2.4 Comparing reasons for TD non-prevention between agile, hybrid, and traditional process models

Figure 9 (A) shows the RBO result considering the lists of coding-related reasons for TD non-prevention of agile and hybrid process models. We did not consider traditional process models because their practitioners did not mention any reason for TD non-prevention. Analyzing the figure, we can see that the similarity level is 10-30%, indicating that agile and hybrid did not share the same vision on reasons for TD non-prevention. This low similarity level is also perceived when we compared the list of reasons for TD non-prevention, as shown in **Table 20**.

Table 20. Top 5 most cited reasons for TD non-prevention related to coding issues per process model.

	Agile	Hybrid
1	**Lack of technical knowledge (2)**	Lack of good technical solutions (2)
2	**Lack of concern about maintainability (1)**	Continuous change of coding standards (1)
3	-	**Lack of concern about maintainability (1)**
4	-	**Lack of technical knowledge (1)**

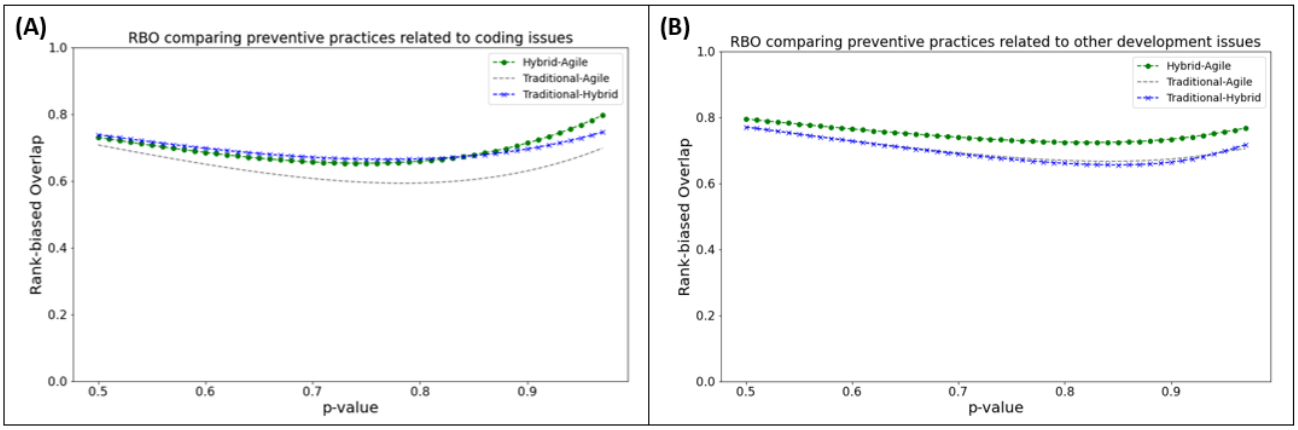

Figure 8. RBO comparing preventive practices related to (A) coding and (B) other software development issues.

About the reasons for TD non-prevention related to other software development issues, **Figure 9 (B)** shows that the similarity level is about 80-90% for the most cited reasons in agile and hybrid process models. But this value decreases, reaching about 55%, when considering the full list of reasons.

Traditional process models did not share the same view on reasons for TD non-prevention as the similarity level is about 30-50%. This low similarity level can be perceived when we analyze the five most cited reasons for TD non-prevention (**Table 21**).

In conclusion, agile and hybrid process models did not share the same vision on coding-related reasons for TD non-prevention, but these models have the same view on the most cited non-coding-related reasons. Traditional process models did not share the same non-coding-related reasons with agile and hybrid process models.

5.2.5 Comparing TD repayment practices between agile, hybrid, and traditional process models

Figure 10 (A) and **Table 22** show the RBO result considering the lists of repayment practices related to coding for each process model. We can see that agile, hybrid, and traditional process models share the same view in repayment practices. The similarity level varies between 80-90%.

Table 21. Top 5 most cited reasons for TD non-prevention related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Short deadline (7)	Short deadline (5)	**Pressure for results (2)**
2	Ineffective management (3)	Ineffective management (3)	Short deadline (2)
3	Lack of predictability in the software development (3)	Lack of predictability in the software development (2)	**Ineffective management (1)**
4	Requirements change (3)	**Legacy system difficult to heal (2)**	**Lack of process maturity (1)**
5	**Architectural evolution (1)**	Requirements change (2)	-

Concerning the repayment practices related to other software development issues, **Figure 10 (B)** shows the comparison for the three process models. Agile and hybrid process models have used almost the same practices (similarity level is about 80-90%). On the contrary, the similarity level when comparing traditional process model with the other two is slightly low, almost 70-80%, for the top 5 ranked elements of their lists as noticed in **Table 23**.

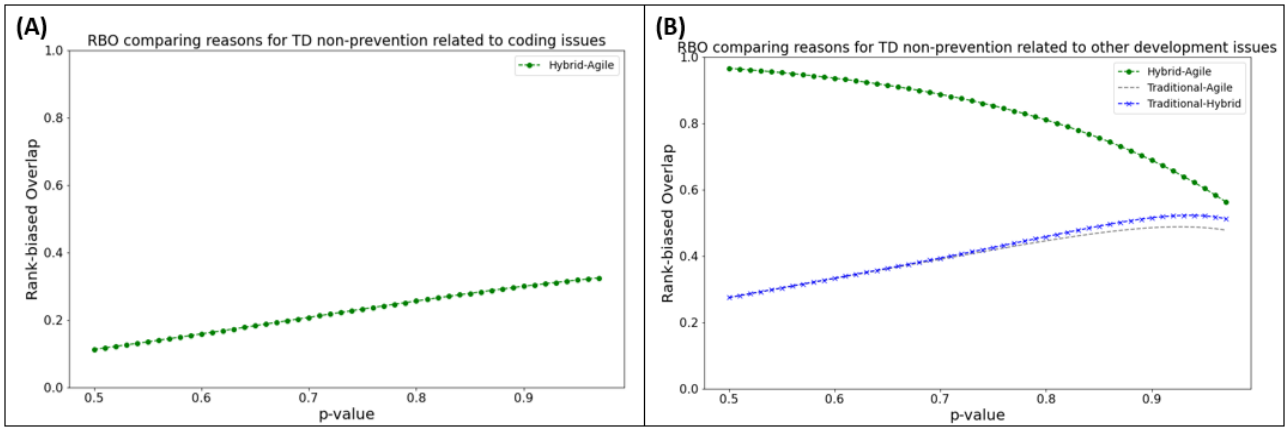

Figure 9. RBO comparing reasons for TD non-prevention related to (A) coding and (B) other software development issues.

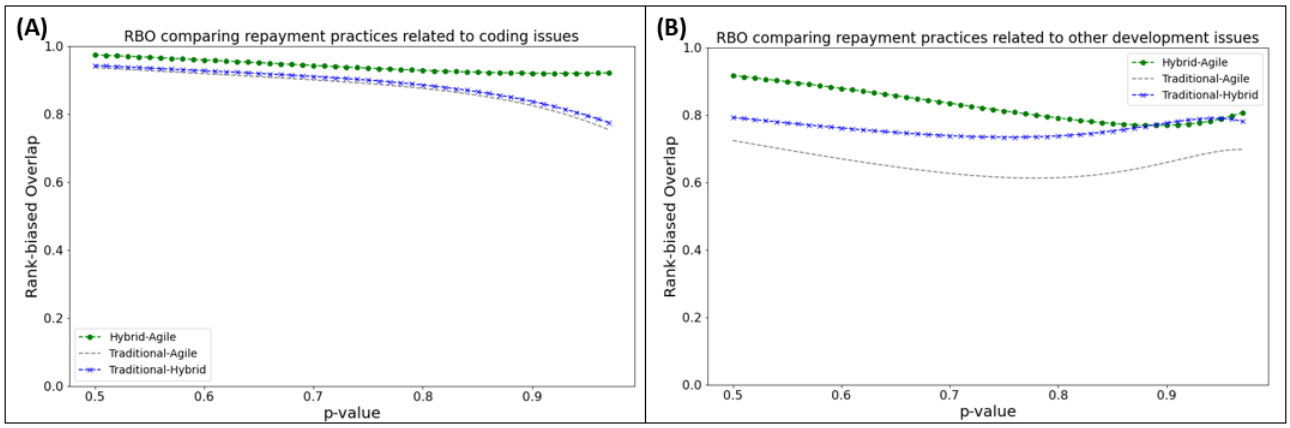

Figure 10. RBO comparing repayment practices related to (A) coding and (B) other software development issues.

Table 22. Top 5 most cited repayment practices related to coding issues per process model.

	Agile	Hybrid	Traditional
1	Code Refactoring (38)	Code Refactoring (37)	Code Refactoring (5)
2	Design Refactoring (14)	Design Refactoring (7)	Design Refactoring (4)
3	Adoption of Good Practices (6)	Adoption of Good Practices (4)	Bug Fixing (1)
4	Solving Tech. Issues (6)	Bug Fixing (3)	Solving Tech. Issues (1)
5	**Code Reviewing (3)**	Solving Tech. Issues (2)	-

Table 23. Top 5 most cited repayment practices related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Investing Effort on TD Repayment Activities (13)	Investing Effort on TD Repayment Activities (16)	Investing Effort on TD Repayment Activities (4)
2	Investing Effort on Testing Activities (12)	Investing Effort on Testing Activities (7)	**Increasing the Project Budget (4)**
3	Prioritizing TD Items (9)	Negotiating Deadline Extension (6)	Negotiating Deadline Extension (4)
4	**Using short Feedback Iterations (5)**	Prioritizing TD Items (6)	Investing Effort on Testing Activities (3)
5	**Implementing Preventive Actions for Avoiding TD(4)**	**Changing Project Scope (4)**	**Update System Documentation (3)**

Practitioners using agile, hybrid, and traditional process models have shared almost the same experience on repayment practices related to coding, but this scenario is different for repayment practices related to other software development issues when considering the context of traditional process models.

5.2.6 Comparing reasons for TD non-repayment between agile, hybrid, and traditional process models

Figure 11 presents the RBO result considering the lists of non-coding-related reasons for TD non-repayment. We did not perform the analysis for coding-related reasons for TD non-repayment because only one reason (*lack of access on component code*) was cited by the participants. Analyzing the figure, we can see that the similarity level is around 80-90%, indicating that practitioners have same view on non-coding-related reasons for TD non-repayment.

In **Table 24**, we can observe that the reasons *focusing on short term goal* and *lack of organizational interest* were the most used reasons for explaining the TD non-repayment. Besides, the other reasons are also very similar among the process models.

In summary, practitioners using agile, hybrid, and traditional process models share the same view on non-coding-related reasons for TD non-repayment.

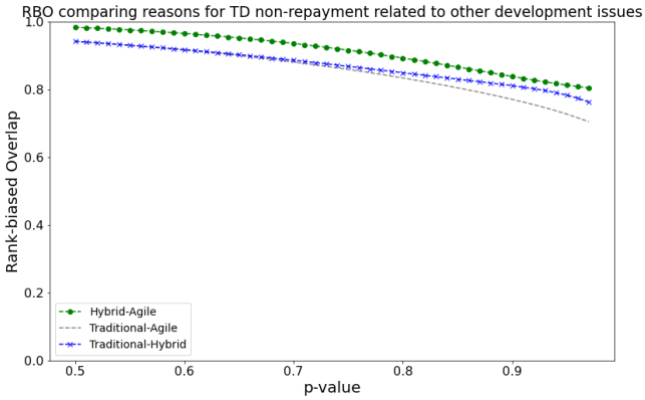

Figure 11. RBO comparing reasons for TD non-repayment related to other software development issues.

Table 24. Top 5 most cited reasons for TD non-repayment related to other development issues per process model.

	Agile	Hybrid	Traditional
1	Focusing on Short Term Goals (28)	Focusing on Short Term Goals (32)	Focusing on Short Term Goals (9)
2	Lack of Organizational Interest (20)	Lack of Organizational Interest (21)	Lack of Organizational Interest (7)
3	Lack of Time (16)	Lack of Time (20)	Cost (5)
4	Cost (13)	Cost (16)	Lack of Time (5)
5	**Effort (7)**	**Lack of Resources (13)**	**Lack of Technical knowledge (3)**

6 Discussion

This section presents an overview of the findings and discusses their implications for practitioners and researchers.

6.1 Summary of findings

The results indicate that coding issues related to the causes, effects, prevention, non-prevention, repayment, and non-repayment of TD are only a small part of the concerns that practitioners face in the presence of TD. Indeed, TD has been more commonly found in other software development issues.

The radar graph presented in **Figure 12** shows the percentages of the distribution of the participants' responses to each of the investigated elements concerning the categories *coding issues* and *other software development issues*. For every investigated element, most of the responses are related to other software development issues. The difference is quite bigger for the elements: causes, prevention, reasons for not preventing, and reasons for not repaying. The values for TD repayment are very close between the two groups (56% vs 44%). This is an indication that, although practitioners perceive that TD is ubiquitous in

software development projects, they also see that its repayment is commonly related to coding issues.

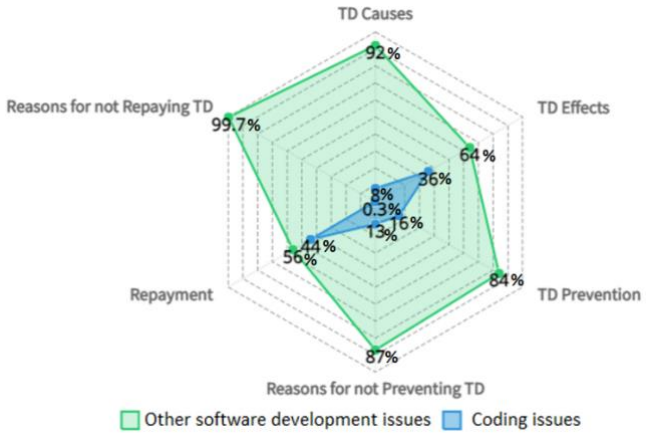

Figure 12. Distribution of the participants' answers on the TD management elements.

We organized the TD management elements into categories. The category *planning and management* concentrated the biggest number of citations of causes, effects, preventive practices, reasons for TD non-prevention, and reasons for TD non-repayment. Alternatively, the category *TD management* has the biggest quantity of preventive practices citations.

All identified categories of each TD management element were represented in a hump diagram. By analyzing the diagram, practitioners can perceive the influence of each TD management element in a specific issue associated with the software development process. These issues correspond to the categories defined in this study. Besides, practitioners can specialize the diagrams following their project context. For illustrating it, we specialized the hump diagram for agile, hybrid, and traditional process models, and compared them with each other.

From the comparison, we noticed that agile and hybrid process models share the same point of view on the TD management elements analyzed in this work. On the other hand, practitioners who adopted traditional process models tend to have a different view on these elements. Strategies defined to support TD management initiatives must consider the specificities of each process model.

6.2 Implications for researchers and practitioners

The hump diagram can guide practitioners, showing how each software development issue is related to each TD management element. Having this information, practitioners can define strategies to mitigate causes, effects, reasons for TD non-prevention, or reasons for TD non-repayment. Also, the combined use of the hump diagram and the detailed results, presented in Section 4 and available at https://bit.ly/37BopIF, provides a comprehensive guidance for software development teams about what to expect from the presence of TD and how to react to them considering several software development issues. For example, practitioners can diagnose the causes of TD by consulting the hump diagram. As the causes from the category *planning and management* are more common in agile software projects, if an agile team has defined preventive practices for these causes and it still identifies new causes, by analyzing the diagram, the team can focus on other causes from more common categories in the agile process, such as *human factors*. Practitioners can also identify preventive practices to avoid TD items in their projects. Suppose a traditional team has applied all preventive practices from the category *planning and management* (with the highest concentration of practices), but the team still felt the effects of TD. The team can apply preventive practices from other categories by analyzing the hump diagram, such as *requirement engineering* and *verification, validation, and test*.

For researchers, our results point out the need of investing more research effort on other issues of the software development. For example, complementary to understanding TD at the code level, it is also necessary to investigate strategies to mitigate the managerial reasons that lead software teams to not repay debt items. Another promising topic for investigation would be the relationship between human factors of the software development and TD.

For practitioners and researchers, the results of RBO analyses bring to the fore the need to further investigate practitioners' perceptions of the elements of TDM. This investigation may reveal differences that can be used to develop methods, techniques, and tools more suited to professionals needs. For example, our findings reveal that agile and traditional processes consider TD prevention differently. Before developing a TD prevention strategy, researchers may investigate agile software development characteristics that influence TD prevention. Also, agile practitioners can learn from traditional practitioners by identifying the differences in perceptions concerning TD prevention.

7 Threats to Validity

As in any empirical study, there are threats to validity in this work. We attempt to remove them when possible, and mitigate their effects when removal is not possible

The main threat to the validity of the conclusion is related to the coding process, as it is a creative process. To mitigate it, the analyses were carried out separately by two researchers, and the consensus was carried out by a third, more experienced one. Also, additional procedures were considered for seeking consistency in the nomenclature used by each replication team during their coding activities. Lastly, the classification of the coded TD management elements into code/non-code, as well as the definition of their categories, are essentially subjective tasks. To mitigate them, we followed a rigorous analysis procedure. The classification process was always performed individually by two researchers, being reviewed by at least one experienced researcher.

Another threat is related to the specialization of hump diagrams per process model. To this end, we relied on the responses from participants to the questions Q8 of the InsighTD questionnaire, which explicitly states the definition of the three categories of processes considered in this research (agile, hybrid, and traditional).

The questionnaire was designed to eliminate threats to internal validity. As discussed in (Rios *et al*., 2020), the questionnaire went through a series of validations (three internal and one external) and a pilot study to identify any issues before its execution. It is also worth mentioning that the participants could act differently from what they usually do because they are part of a study. To avoid this, we clearly explain the purpose of the study and ask participants to answer the questions based on their own experience. We also state explicitly that the questionnaire is anonymous, and that the data collected is analyzed without considering the identity of the participants. Also, participants may have misinterpreted the use of the terms prevention and repayment of TD. To investigate whether this threat manifested, all responses on how participants avoided and repaid the debt item were analyzed (Q23 and Q27) to analyze if there were invalid answers. A high proportion of invalid responses would mean that the questions could be misinterpreted. In the end, we did not identify any invalid response, indicating that this threat did not appear in the study.

Lastly, external validity threats were reduced by targeting industry professionals and seeking to achieve participant diversity among survey respondents. In search of more generalizable results, InsighTD is being replicated in other countries.

8 Concluding Remarks

In this paper, we investigate the relation between TD management elements (causes, effects, preventive practices, repayment practices, reasons for TD non-prevention, and reasons for TD non-repayment) and software development issues related to coding or other activities. Also, we categorize these elements and organize them into hump diagrams. Further, we define a hump diagram for each process model (agile, hybrid, and traditional) to demonstrate how the diagram can be specialized by practitioners following one of their project's variables, such as, process model and role.

The next steps of this work include (i) to investigate whether the type of debt impacts how practitioners see TD management elements, (ii) to develop a TD management instrument encompassing the hump diagram and the detailed results, and (iii) to empirically assess this instrument on the supporting of TD management. We also intend to investigate the main human factors associated with TD.

Acknowledgements

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) Finance Code 001 and the Conselho Nacional de Desenvolvimento Científico e Tecnológico – CNPq. This research was also supported in part by funds received from the David A. Wilson Award for Excellence in Teaching and Learning, which was created by the Laureate International Universities network to support research focused on teaching and learning.

References

Alves, N.S.R., Mendes, T.S., Mendonça, M.G., Spínola, R., Shull, F., & Seaman, C. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70, 100-121. DOI: https://doi.org/10.1016/j.infsof.2015.10.008.

Berenguer, C., Borges, A., Freire, S., Rios, N., Tausan, N., Ramac, R., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., López, G., Falessi, D., Seaman, C., Mandic, V., Izurieta, C., & Spínola, R. (2021). Technical Debt is not Only about Code and We Need to be Aware about It. In *Proceedings of the XX Brazilian Symposium on Software Quality (SBQS '21)*. ACM, New York, NY, USA, 1–12. DOI: https://doi.org/10.1145/3493244.3493285.

Besker, T., Ghanbari, H., Martini, A., & Bosch, J. (2020). The influence of technical debt on software developer morale. *Journal of Systems and Software*, 167. DOI: https://doi.org/10.1016/j.jss.2020.110586.

Cunningham, W. (1992). The WyCash portfolio management system. *ACM SIGPLAN OOPS Messenger*, 4, 2 (April 1993), 29-30. DOI: https://doi.org/10.1145/157710.157715.

Freire, S., Rios, N., Mendonça, M., Falessi, D., Seaman, C., Izurieta, C., & Spínola, R. (2020a). Actions and impediments for technical debt prevention: results from a global family of industrial surveys. In *Proceedings of the 35th ACM/SIGAPP Symposium on Applied Computing*, Brno, 1548–1555.

Freire, S., Rios, N., Gutierrez, B., Torres, D., Mendonça, M., Izurieta, C., Seaman, C., & Spínola, R. (2020b). Surveying Software Practitioners on Technical Debt Payment Practices and Reasons for not Paying off Debt Items. In *Proceedings of the Evaluation and Assessment in Software Engineering*. Trondheim, 210–219.

Freire, S., Rios, N., Perez, B., Castellanos, C., Correal, D., Ramac, R., Mandic, V., Tausan, N., Pacheco, A., López, G., Mendonça, M., Izurieta, C., Falessi, D., Seaman, C., & Spínola, R. (2021a). Pitfalls and Solutions for Technical Debt Management in Agile Software Projects. *IEEE Software*, vol. 38, no. 6, pp. 42-49, Nov.-Dec. 2021. DOI: 10.1109/MS.2021.3101990.

Freire, S., Rios, N., Perez, B., Castellanos, C., Correal, D., Ramac, R., Mandic, V., Tausan, N., López, G., Pacheco, A., Falessi, D., Mendonça, M., Izurieta, C., Seaman, C., & Spínola, R. (2021b). How Experience Impacts Practitioners' Perception of Causes and Effects of Technical Debt. In *Proceedings of the IEEE/ACM 13th International Workshop on Cooperative and Human Aspects of Software Engineering* (CHASE). DOI: 10.1109/CHASE52884.2021.00011.

Freire, S., Rios, N., Pérez, B., Correal, D., Mendonça, M., Izurieta, C., Seaman, C., & Spínola, R. (2021c). How do technical debt payment practices relate to the effects of the presence of debt items in software projects? In *Proceedings of the IEEE International Conference on Software Analysis, Evolution and Reengineering* (SANER). DOI: 10.1109/SANER50967.2021.00074.

Guo, Y., Spínola, R.O., & Seaman, C. (2016). Exploring the costs of technical debt management --- a case study. *Empirical Software Engineering*, 21, 1 (February 2016), 159–182. DOI: https://doi.org/10.1007/s10664-014-9351-7.

Izurieta, C., Vetrò, A., Zazworka, N., Cai, Y., Seaman, C., & Shull, F. (2012). Organizing the technical debt landscape. In *Proceedings of the 3rd International Workshop on Managing Technical Debt* (MTD). Zurich, 23-26. DOI: https://doi.org/10.1109/MTD.2012.6225995.

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

Li, Z., Avgeriou, P., & Liang, P. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101, 193–220. DOI: https://doi.org/10.1016/j.jss.2014.12.027.

Lim, E., Taksande, N., & Seaman, C. (2012). A balancing act: What software practitioners have to say about technical debt. *IEEE Software*, 29, 6 (November 2012), 22–27. DOI: https://doi.org/10.1109/MS.2012.130.

Martini, A., Stray, V., & Moe, N.B. (2019). Technical-, social-and process debt in large-scale agile: an exploratory case-study. In *Proceeding of the International Conference on Agile Software Development* (pp. 112-119). Springer, Cham.

Ramač, R., Mandić, V., Taušan, N., Rios, N., Freire, S., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., Lopez, G., Izurieta, C., Seaman, C., & Spinola, R. (2022). Prevalence, common causes and effects of technical debt: Results from a family of surveys with the IT industry. *Journal of Systems and Software*, 184, 111114. DOI: https://doi.org/10.1016/j.jss.2021.111114.

Ribeiro, L.F., Farias, M.A.F, Mendonça, M., & Spínola, R.O. (2016). Decision criteria for the payment of technical debt in software projects: A systematic mapping study. In *Proceedings of the 18th International Conference on Enterprise Information Systems (ICEIS)*. DOI: https://doi.org/10.5220/0005914605720579

Rios, N., Freire, S., Pérez, B., Castellanos, C., Correal, D., Mendonça, M., Falessi, D., Izurieta, C., Seaman, C., & Spínola, R. (2021). On the Relationship Between Technical Debt Management and Process Models. *IEEE Software*.

Rios, N., Mendonça, M., & Spínola, R. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology*, 102, 117-145. DOI: https://doi.org/10.1016/j.infsof.2018.05.010.

Rios, N., Spínola, R.O., Mendonça, M., & Seaman, C. (2019). Supporting analysis of technical debt causes and effects with cross-company probabilistic cause-effect diagrams. In *Proceedings of the IEEE/ACM International Conference on Technical Debt (TechDebt)*. DOI: https://doi.org/10.1109/TechDebt.2019.00009.

Rios, N., Spínola, R.O., Mendonça, M., & Seaman, C. (2020). The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. *Empirical Software Engineering*, 25, 3216-3287.

Saraiva, D., Neto, J. G., Kulesza, U., Freitas, G., Reboucas, R., & Coelho, R. (2021). Technical Debt Tools: A Systematic Mapping Study. In *Proceedings of the 23rd International Conference on Enterprise Information Systems*. DOI:10.5220/0010459100880098.

Strauss, A. & Corbin, J. (1998). Basics of qualitative research: Techniques and procedures for developing grounded theory. *Sage Publications*.

Tamburri, D.A., Kruchten, P., Lago, P. & van Vliet, H. (2015). Social debt in software engineering: insights from industry. *Journal of Internet Services and Applications*, 6(1), 1-17.

Webber, W., Moffat, A., & Zobel, J. (2010). A Similarity Measure for Indefinite Rankings. *ACM Transactions on Information Systems*, Vol. 28, no.4.

Wohlin, C., Runeson, P., Host, M., Ohlsson, M.C., Regnell, B., & Wesslen, A. (2012). Experimentation in software engineering: An introduction. *Springer*.

Zazworka, N., Vetro', A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., & Shull, F. (2014). Comparing four approaches for technical debt identification. *Software Quality Journal*, 22, 403–426 (2014). DOI: https://doi.org/10.1007/s11219-013-9200-8.