Actions and Impediments for Technical Debt Prevention: Results from a Global Family of Industrial Surveys

Sávio Freire
Federal University of Bahia
Federal Institute of Ceará
Brazil
savio.freire@ifce.edu.br

Nicolli Rios
Federal University of Bahia
Brazil
nicollirioss@gmail.com

Manoel Mendonça
Federal University of Bahia
Brazil
manoel.mendonca@dcc.ufba.br

Davide Falessi
California Polytechnic State University United States
dfalessi@calpoly.edu

Carolyn Seaman
Univ. of Maryland Baltimore County
United States
cseaman@umbc.edu.org

Clemente Izurieta
Montana State University
United States
clemente.izurieta@montana.edu

Rodrigo O. Spínola
Salvador University
State University of Bahia
Brazil
rodrigo.spinola@unifacs.br

ABSTRACT

Background: Preventing the occurrence of technical debt (TD) in software projects can be cheaper than its payment. Prevention practices also help in catching inexperienced developers' 'not-so-good' solutions. However, little is known on how to prevent the occurrence of TD. **Aims**: To investigate, from the point of view of software practitioners, preventive actions that can be used to curb the occurrence of TD and the impediments that hamper the use of those actions. **Method**: We use data from the *InsighTD* Project, a family of industrial surveys specifically designed to study software engineering TD. We use a corpus of answers from 207 practitioners across different geographic locations to identify and analyze – both quantitatively and qualitatively – the TD preventive actions most used in practice. **Results**: We found that *project planning*, *adoption of good practices*, *well-defined requirements*, *creating tests*, and *training* are the most cited preventive actions that curb TD in software projects. We also identified seven preventive action categories and defined relationships among them and TD types. On the other hand, the main impediments to prevent TD are related to inappropriate project planning and lack of expertise of the team. **Conclusions**: Our list of preventive actions and impediments can help practitioners to implement policies for the sector and guide TD researches in a problem-driven way.

CCS CONCEPTS

• **General and reference** → **Empirical studies** • **Software and its engineering** → Maintaining software

SAC '20, March 30-April 3, 2020, Brno, Czech Republic
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6866-7/20/03…$15.00
DOI: https://doi.org/10.1145/3341105.3373912

KEYWORDS

Technical debt, technical debt management, technical debt prevention, preventive actions, survey, InsighTD

ACM Reference format:

S. Freire, N. Rios, M. Mendonça, D. Falessi, C. Seaman, C. Izurieta, and R. O. Spínola. 2020. In *Proceedings of ACM SAC Conference, Brno, Czech Republic, March 30- April 3, 2020 (SAC'20)*, 8 pages. DOI: https://doi.org/10.1145/3341105.3373912

1 INTRODUCTION

Technical debt (TD) describes the effects of developing immature artifacts during software projects, bringing benefits in the short term, but risking high payment with interest later in the project life cycle. The benefits can be observed as higher productivity and lower costs, while "interest" is associated with unexpected delays in system evolution and difficulty in achieving quality criteria defined for the project [1, 2]. TD can be a good investment as long as the project team knows about its presence and the increased risks it imposes on the project [3]. If properly managed, TD can help the project achieve its goals sooner or more cheaply. On the other hand, if debt items are unmanaged, they can cause financial and technical problems, increasing software maintenance and evolution costs, leading to a situation where the whole future of the software is jeopardized [4, 5].

Several research articles have addressed TD, seeking to identify strategies, tools, and activities for its management [6, 7]. However, *little has been published about how to prevent the occurrence of TD in software projects* [6]. This issue deserves investigation because it is fair to expect that TD prevention can sometimes be "cheaper" than its repayment. Moreover, prevention may also help other TD management activities. For example, setting up prevention practices helps in catching inexperienced developers' 'not-so-good' solutions [8].

A preventive action is an intentional activity, aligned with the project management plan, that ensures the future performance of

the project work [9]. When applied to TD, preventive actions can support the development team in applying good practices that minimize the occurrence of debt.

The goal of this work is to investigate, from the point of view of software practitioners, the preventive actions that can be used to avoid the occurrence of TD and the impediments that hamper the application of these actions. This work uses data collected by the *InsighTD* Project [2], which is a globally distributed family of industrial surveys on the causes and effects of TD [10]. A total of 207 professionals from the Brazilian and North American software industry responded the first round of surveys. This work analyzes this data through qualitative and quantitative strategies. First, it characterizes the study participants. Then, it identifies the participants that indicated that it was possible to prevent the occurrence of debt, and qualitatively analyzes the preventive actions cited by them. For those who indicated that it was not possible to curb the occurrence of TD, it qualitatively analyzes the possible impediments for TD prevention.

Results show that *following project planning*, *adoption of good practices*, *well-defined requirements*, *creating tests*, and *training* are the most cited preventive actions to minimize the occurrence of TD in software projects. The results also point out seven categories of preventive actions and the relationships among each one and TD types. These relationships provide an indication on how those actions are used to minimize the occurrence of a TD type, giving hints on how to be prepared to and, when necessary, fight against the presence of TD.

Alternatively, the main impediments to prevent TD are related to inappropriate project planning and managing, inappropriate software development process, and lack of expertise and maturity of the team.

This paper has implications for practitioners and researchers. For practitioners, it presents the main preventive actions and impediments faced by software teams, which can be used to support decision making regarding the definition of strategies to minimize the occurrence of debt in projects. For researchers, the results shed new light on how to prevent TD. The results, originating from the software industry, provide a matter-of-fact direction of demands that need to be better understood in the area.

This paper is organized in six other sections. Section 2 presents background about the *InsighTD* project, and TD management and its prevention. Section 3 discusses the research method. Then, Section 4 presents and discusses the results of *InsighTD* concerning TD prevention. The implications of the study for both researchers and practitioners are presented in Section 5. Section 6 discusses the threats to validity. Lastly, Section 7 presents some final remarks and the next steps of this work.

2 BACKGROUND

This section introduces the *InsighTD* Project and TD management concepts related to this work.

[2] http://td-survey.com/

2.1 The *InsighTD* Project

InsighTD is a globally distributed family of industrial surveys initiated in 2017. Planned cooperatively among TD researchers from around the world, the project aims to organize an open and generalizable set of empirical data on the state of practice and industry trends in the TD area. This data includes the causes that lead to TD occurrence, the effects of its existence, how these problems manifest themselves in the software development process, and how software development teams react when they are aware of the presence of debt items in their projects. Its design establishes the foundations for the survey to be continuously replicated in different countries. Up to date, researchers from 11 countries (Brazil, Colombia, Chile, Costa Rica, Finland, India, Italy, Norway, Saudi Arabia, Serbia, and the United States) have joined the project. At the moment, we have concluded data collections of the *InsighTD* replications in Brazil and the United States.

Rios *et al.* [10] discussed the basic survey design and the preliminary results of the first round of *InsighTD*. In that paper, the authors focus on the discussion on the top 10 causes and effects of TD. Rios *et al.* [11] complemented the discussion of the previous work, focusing specifically on the causes and effects of TD in agile software projects. More recently, Rios *et al.* [12] proposed the use of cross-company probabilistic cause-effect diagrams to represent information about the TD causes and effects being analyzed.

Although significant analysis has already been conducted over the available *InsighTD* data, much still remains to be studied. In particular, the data has yet to be analyzed with regards to TD prevention.

2.2 Technical Debt Management and its Prevention

Technical debt management facilitates decision-making about the need to eliminate a debt item and the most appropriate time to do this [13]. If adequately managed, TD can help the project to achieve its goals sooner or more cheaply. Thus, the management of TD focuses on reducing its negative impact, being a decisive factor for the success of software projects.

Li *et al.* [14] conducted a systematic mapping study of the literature to understand the state of the art concerning TD management. The authors list eight activities related to TD management: identification, quantification, prioritization, prevention, monitoring, payment, documentation, and communication. From the results of a tertiary study of the area, Rios *et al*. [6] added the following activities to the previous list: TD visualization, time to market analysis, and scenario analysis. Rios *et al*. [6] also analyze existing strategies and tools that support the implementation of each of these activities. No tool or strategy was identified to support TD prevention activities.

More recently, Rios *et al*. [15] investigated if TD can be prevented, and if, in terms of effort, it is better to prevent debt or incur it and pay it off later on. Through an interview-based case study with ten practitioners, the authors indicate that debt can be prevented, and it is better to work on prevention activities than to pay it off later on.

Thus, although there are already research articles indicating the importance of focusing on the prevention of TD items, little is known about possible TD preventive actions, as well as impediments that may hamper taking these actions in a software organization. These issues are precisely the topic addressed in this paper.

3 RESEARCH METHOD

3.1 Research Questions

In order to achieve our goal, we defined the following main Research Question (RQ) "***How do software development teams prevent technical debt in their projects?***". The goal of this RQ is to identify the main actions that software practitioners can use to prevent the occurrence of TD. To investigate it, we broke down this question into the following sub-questions:

RQ1: *Can development teams prevent the occurrence of TD in software projects?* This question conveys our pre-conception that TD can be prevented. Through it, we will explore practitioners' responses to the *InsighTD* dataset and calculate how often TD items could be prevented.

RQ2: *What are the main preventive actions indicated by software development teams to prevent TD?* This question seeks to investigate the possible kinds of preventive actions and the most commonly cited actions used to minimize the occurrence of debt.

RQ3: *What are the main preventive actions indicated by software development teams to curb the occurrence of each TD type?* This question is aimed at identifying the preventive actions used to prevent each TD type, such as, architecture, code, design, and test debt.

RQ4: *What are the impediments to preventing TD?* The purpose of this question is to investigate the possible impediments that contribute to the non-application of TD preventive actions.

3.2 Data Collection

The data were collected in the context of the *InsighTD* project. The *InsighTD* questionnaire consists of 28 questions. Table 1 presents the subset of the survey's questions related to the context of this work. Q1 to Q8 capture the characterization questions, Q9 and Q10 identify participants' knowledge level on TD, Q13 and Q15 ask participants to provide an example of TD item that occurred in their project (this example would then be used as the basis for answering questions about prevention) and representativeness of this example, respectively, and in Q22, Q23, and Q28, the participants answer questions about TD prevention.

The questionnaire was only sent to practitioners, because the objective of *InsighTD* is to investigate the state of the practice of TD. Some keywords related to software development activities and roles were used in LinkedIn to identify the participants. Also, invitations were sent to industry-affiliated member groups, mailing lists, and industry partners. The same strategy was applied both in Brazil and in the United States.

Table 1: Subset of the *InsighTD* survey's questions related to TD prevention (adapted from [10])

No.	Question (Q)	Type
Q1	What is the size of your company?	Closed
Q2	In which country you are 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
Q9	How familiar you are with the concept of Technical Debt?	Closed
Q10	In your words, how would you define TD?	Open
Q13	Give an example of TD that had a significant impact on the project that you have chosen to tell us about:	Open
Q15	About this example, how representative it is?	Closed
Q22	Do you think it would be possible to prevent the type of debt you described in question 13?	Closed
Q23	If yes, how? If not, why?	Open
Q28	Considering your personal experience with TD management, what actions have you performed to prevent its occurrence?	Open

3.3 Data Analysis

Because the questionnaire is composed of closed and open questions, the work adopted several data analysis procedures. For closed questions, we used descriptive statistics to get a better understanding of the data. To verify the central tendency of the ordinal and interval data, we used the mode and median statistics. By calculating the share of participants choosing each option, we could analyze the nominal data. This data analysis procedures support responding to RQ1.

To analyze the open questions on preventive actions for TD, we applied qualitative data analysis techniques [16, 17]. Based on the answers given to Q23 and Q28, we followed an inductive logic approach. We applied manual open coding to responses of Q23 and Q28. Initially, the first author coded the set of all answers for three subsets. Two subsets represented the preventive actions, and they were formed in two different ways: either by the answers to Q23 when the participants' responses were **positive** for Q22 or by these answers and the answers for Q28. Both of the subsets support responding to RQ2. The other subset was composed of the answers to Q23 when the participants' responses were **negative** for Q22, revealing impediments to preventing TD (RQ4). The second author's role was to review all codes. Disagreements were resolved by the last author. Next, we analyzed the extracted codes in both subsets, identifying the codes that had the same meaning. This process resulted in the final list of a set of standard codes. Finally, we derived higher level categories using axial coding [17]. This process was performed iteratively until reaching the state of saturation, i.e., a point where no new codes or categories were identified.

An example of this process is as follows: two participants cited the following ways to prevent TD: "Better planning, better allocation

of technical staff in the project," and "Through better sprint planning and/or time flexibility." The extracted codes were *better planning*, *allocation of technical staff*, *better sprint planning*, and *time flexibility*. As *better planning* and *better sprint planning* are different nomenclature for the same preventive action, we unified them as *well-defined planning*. Immediately following the extraction of codes, we identified the following final list of codes: *well-defined planning*, *allocation of technical staff*, and *time flexibility*. Finally, these codes were categorized as *following a well-defined project planning* because all of them are related to project planning issues.

To answer RQ3, we considered the example given for each participant in Q13. By using a list of TD indicators [18], the second author identified the TD type of each example (Q13). The last author reviewed the obtained results. We then associated the preventive actions coded in Q23 with TD types.

4 RESULTS AND DISCUSSION

The survey was applied in Brazil between December 2017 and January 2018. The replication in the United States occurred between February and April of 2019. In total, 207 professionals from the software industry answered the survey questions (107 from Brazil and 100 from the United States). Only the responses of professionals with previous knowledge on TD were considered in the dataset. We did this filter by analyzing the answers for Q10 and Q13, in which we identified if the understanding of the participant was aligned with the concept of TD considered in the *InsighTD* project [10]. Several participant roles were described (see Table 2), however, the vast majority identified as developers. The roles of project manager, software architect, tester, and requirements analyst also stood out. With regards to the participant's level of experience, we considered the following rank of categories to group the participants: novice, beginner, competent, proficient, and expert. Most participants indicated they were proficient (~36%), followed by expert (~29%), competent (~23%), beginners (~12%), and novice (~1%).

Organizations of different sizes are represented in the dataset. Most of them are medium-sized companies (39%, organizations with 51 to 1000 employees), followed by large (37%, more than 1000 employees) and small (24%, up to 50 employees). Regarding the processes used in the projects, 49% were agile, 39% hybrids, and 12% traditional. Team sizes are varied; however, participants mainly work in teams of 10-20 people (28%). Other team sizes included teams with 5-9 people (26%), less than 5 people (20%), more than 30 people (20%), and 21-30 people (6%). The most common system age was between 2 to 5 years old (36%), followed by 1 to 2 years old (23%), 5 to 10 years old (16%), less than 1-year-old (15%) and more than ten years old (10%). The systems were typically between 10 KLOC and 1 million LOC (55%) in size, but we found systems with less than 10 KLOC (19%), from 1 to 10 million LOC (18%), and more than 10 million LOC (7%) in size.

In summary, the dataset reflects the Brazilian and North American software industry diversity, containing different roles of practitioners and levels of experience, organizations with different sizes, and projects with different age, size, team size, and process models.

Table 2: Participant Roles

Role	#	%
Developer	100	48.1%
Project Leader / Project Manager	31	15%
Software Architect	27	13%
Test Manager / Tester	18	8.7%
Requirements Analyst	11	5.3%
Process Analyst	4	2.4%
Database Administrator	3	1.4%
Infrastructure Analyst	3	1.4%
Performs multiple functions	3	1.4%
Business Analyst	2	1%
Configuration Manager	2	1%
Quality Analyst	2	1%
Data Scientist	1	0.5%

Caption:
- Quantity of participant roles
% - Percentage of participant roles

4.1 Can development teams prevent the occurrence of TD in software projects? (RQ1)

Initially, we asked (Q22) whether it would be possible to prevent the specific TD item described by the participant in question Q13. Of the total, 184 (88%) of participants indicated that it would be possible. This result is a relevant percentage since the majority of participants (82%) indicated (in Q15) that their example was relevant, and is a kind of situation that happens very often or occurs from time to time in the project.

4.2 What are the main preventive actions indicated by software development teams to prevent TD? (RQ2)

Fig. 1 shows the top 10 most commonly cited preventive actions for TD from a total of 136 identified actions, as informed by the 207 participants in Q23 and Q28. The complete list of preventive actions is publicly available at the following URL: http://bit.ly/35hyswB. We can observe that *following the project planning*, *adoption of good practices*, *well-defined requirement*, *creating tests*, and *training* are the most cited preventive actions that are used to minimize the occurrence of TD. These actions

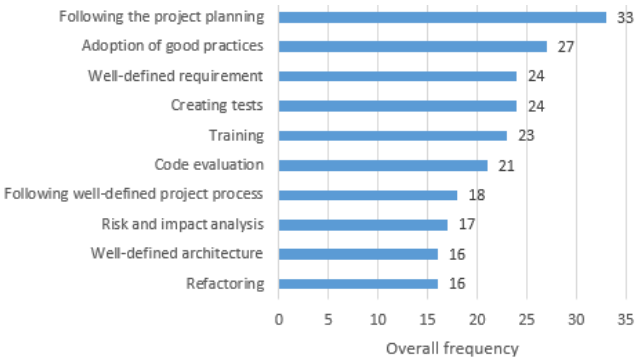

Figure 1: Top 10 cited preventive actions for TD

were cited by almost 26% of the participants. *Code evaluation*, *following well-defined project process*, *risk and impact analysis*, *well-defined documentation*, and *refactoring* are other well cited preventive actions reported by at least 17% of the participants.

When we look at the top 10 preventive actions, three sets of actions stand out. In the first one, *adoption of good practices*, *well-defined architecture*, *refactoring*, *code evaluation*, *well-defined requirements*, and *creating tests* are related to quality control and software development. The actions *following the project planning*, *risk and impact analysis*, and *following well-defined project process* are associated with *project planning and process*. Lastly, the improvement of the technical level of the team is also commonly cited through the action *training.* The top 10 preventive actions correspond to ~44% of the overall frequency of citations.

From the grouping of all 136 actions, we identified the following seven categories of preventive actions:

- **Following a well-defined project planning**: is related to the actions associated with project planning activities. Among them, we highlight *following the project planning*, *risk and impact analysis*, *well-planned deadlines*, *effective monitoring*, and *appropriate tasks allocation;*
- **Adopting of good practices for software development**: includes the actions related to software development activities, such as *adoption of good practices*, *well-defined documentation*, *well-defined architecture*, *project design*, *appropriate use of design patterns*, and *use the most appropriate version of the technology;*
- **Having an effective team**: encompasses the actions that improve the technical knowledge and the motivation of the team. Among them, we highlight *training*, *good communication on the team*, *good allocation of resources in the team*, *readiness of team*, and *focus*;
- **Controlling and measuring the quality in the project**: groups actions associated to quality assurance, such as: *creating tests*, *code evaluation*, *refactoring*, *creating automated tests*, and *code standardization*;
- **Controlling and negotiating the software requirements**: includes actions related to requirements engineering activities. Among them, we highlight *well-defined requirement*, *good communication between stakeholders*, *well-defined scope statement*, *requirements change tracking* and *customer commitment*;
- **Following and improving a well-defined process**: organizes the actions associated to process and its management, such as: *following well-defined project process*, *iterative process*, *flexibilization in the defined process*, *improving software development process*, and *understanding the development process followed by the team*;
- **Identifying, managing, and estimating TD**: contains the actions applied to the TD management. Among them, we highlight *implementation of a TD payment strategy*, *TD monitoring*, *implementation of a TD identification strategy*, and *prioritization of TD payment*.

Table 3 shows the identified categories, reporting the number of preventive actions cited without repetition (**#PA**) and the total number (i.e., count) of actions (**#CA**) cited in each category. Column **%CA** indicates the percentage of #CA in relation to the total of all cited actions. We can observe that the three most cited categories by survey participants represent 64% of the total citations, indicating that those categories play a central role in TD prevention initiatives. The categories *having an effective team* and *controlling and negotiating the software requirements* were also commonly remembered by the practitioners as being relevant when preventing TD.

The category *identifying, managing, and estimating TD*, corresponds to only 6%, indicating that few participants explicitly think of the management of TD as a preventive action in it of itself. This result is expected since most of the TD management activities occur after the debt is already inserted in the project. Finally, despite the fact that the category *following and improving a well-defined process* has only been cited by 6% of the participants, almost all citations in this category refer to the preventive action *following a well-defined project process.*

4.3 What are the main preventive actions indicated by software development teams to curb the occurrence of each TD type? (RQ3)

To answer the RQ3, we analyzed two relationships: between TD types and preventive actions, and between TD types and preventive action categories. The first analysis reveals a list of preventive actions by type of debt while a list of preventive action categories by TD type is evidenced in the second analysis.

4.3.1 Preventive Action by TD Type. Table 4 shows the identified preventive actions that have the highest number of relationships with TD types. The complete data is available at http://bit.ly/2LN9JbS. In this table, we report the number of technical debt types (**#TDT**) that are related to each preventive action. Higher #TDT indicates that a preventive action can have a broad impact in terms of preventing several types of debt.

Through quantitative analysis, we can notice that the preventive action *following the project planning* is related to nine TD types, while the preventive actions *following well-defined project*

Table 3: Relationship between Categories and Preventive Actions

Category	#PA	#CA	%CA
Adopting of good practices for software development	31	114	23%
Controlling and measuring the quality in the project	28	107	21%
Following a well-defined project planning	21	99	20%
Having an effective team	26	72	14%
Controlling and negotiating the software requirements	9	47	9%
Identifying, managing, and estimating TD	12	31	6%
Following and improving a well-defined process	9	29	6%

Caption:
#PA - Number of preventive actions cited without repetition.
#CA - Count of actions cited in each category.
%CA - Percentage of #CA in relation to the total of all cited actions.

Table 4: Relationship between Preventive Actions and TD Types

Preventive Action	#TDT
Following the project planning*	9
Following well-defined project process	7
Training	7
Good communication between stakeholders	6
Adoption of good practices	5
Risk and impact analysis	5
Well-defined requirement	5
Implementation of a TD payment strategy	4
Project design	4
Quality control	4
Readiness of team	4
Refactoring	4
TD monitoring	4
Well planned deadlines	4
Well-defined architecture	4
Well-defined documentation	4

Caption:
#TDT - Number of TD types associated with the preventive action.
* Preventive actions in common with Fig. 1 are highlighted in bold.

process and *training* are related to seven types. By looking at Table 4, we can also observe that the first seven positions could help the prevention of at least five different TD types. Comparing the preventive actions presented in Table 4 to ones presented in Fig.1, only the preventive actions *creating tests* and *code evaluation* are not common in both. This is an indication that, despite them being commonly considered by development teams, they have limited impact in terms of the number of types of debt that could be prevented by using them

Table 5 details the most commonly cited preventive actions and their respective types of debt. The complete data is available at http://bit.ly/2Eav0I8. The quantity of relationships among a TD type and a preventive action is showed in parentheses. For instance, we found the preventive action *following well-defined project process* was related to *architecture debt* four times in our analysis. We can observe that of the 15 types of TD evidenced by Rios *et al.* [6], only *people debt* and *build debt* do not have any associated preventive action. Except *service debt* and *versioning debt*, TD types are associated with at least one preventive action (highlighted in bold) contained in the Top 10 presented in Fig. 1.

4.3.2 Preventive Action Category by TD Type. Table 6 presents the relationship between preventive action categories and TD types. To create this relationship, we considered that if a preventive action of a category was related to a type of debt, then this category would also be related to that type. We can observe that each category of preventive action is related to at least seven types of TD. Therefore, combining preventive actions from different categories can be a good strategy for curbing the presence of different TD items at the same time.

Table 7 details the relation of each TD type to each category, reporting the number of times (#) that a TD type was found in each category. We used acronyms to identify each preventive action category, such as **AD** for the category *adopting of good practices for software development*, **CP** for the category *controlling and measuring the quality in the project*, **FP** for the

Table 5: Most cited Preventive Action for TD Type

TD Type	Preventive Action
Architecture Debt	- **Following well-defined project process** (4)*
	- **Following the project planning** (3)
	- **Well-defined architecture** (3)
	- **Training** (2)
	- Use the most appropriate version of the technology (2)
Code Debt	- **Well-defined requirement** (4)
	- **Code evaluation** (3)
	- Code standardization (3)
	- **Adoption of good practices** (2)
	- **Following the project planning** (2)
Debt Automation	- Creating automated tests (1)
Test Debt	- **Following the project planning** (1)
	- Implementation of a TD payment strategy (1)
	- Task automatization (1)
Defect Debt	- **Training** (1)
	- Version control (1)
	- **Well-defined architecture** (1)
Design Debt	- **Following the project planning** (2)
	- TD monitoring (2)
	- **Well-defined architecture** (2)
	- **Well-defined requirement** (1)
	- **Risk and impact analysis** (1)
Documentation Debt	- Well-defined documentation (5)
	- **Following the project planning** (4)
	- Appropriate tasks allocation (2)
	- **Training** (2)
	- **Well-defined requirement** (2)
Infrastructure Debt	- **Following the project planning** (2)
	- **Adoption of good practices** (1)
	- Focus (1)
	- **Refactoring** (1)
	- Using Components (1)
Process Debt	- **Following the project planning** (6)
	- **Following well-defined project process** (3)
	- **Refactoring** (3)
	- **Risk and impact analysis** (3)
	- **Well-defined requirement** (3)
Requirement Debt	- **Well-defined requirement** (8)
	- **Following the project planning** (5)
	- **Following well-defined project process** (2)
	- Good allocation of resources in the team (2)
	- Scope statement (2)
Service Debt	- Following architectural pattern (1)
	- Framework update (1)
Test Debt	- **Creating tests** (7)
	- **Following the project planning** (4)
	- **Refactoring** (3)
	- **Risk and impact analysis** (3)
	- **Training** (3)
Usability Debt	- **Adoption of good practices** (1)
Versioning Debt	- Organizing code repository (1)
	- Using continuous integration (1)

***** The number in parentheses represents the quantity of relationships among a TD type and a preventive action.

Table 6: Relationship between Preventive Action Categories and TD Types

Category	#NT
Adopting of good practices for software development	12
Controlling and measuring the quality in the project	10
Following a well-defined project planning	9
Having an effective team	9
Identifying, managing, and estimating TD	8
Controlling and negotiating the software requirements	7
Following and improving a well-defined process	7

Caption:
#NTT - Number of TD types cited.

Table 7: Relationship between TD Types and Preventive Action Categories

TD Type	Category of Preventive Action						
	#AD	#CP	#FP	#HT	#CR	#ID	#FW
Process Debt	**14**	7	12	2	5	1	4
Code Debt	8	**11**	8	10	6	2	2
Architecture Debt	**9**	1	4	3	1	1	5
Design Debt	**5**	0	4	2	2	**5**	2
Infrastructure Debt	**2**	1	**2**	1	0	**2**	0
Defect Debt	**2**	0	0	1	0	0	0
Service Debt	**1**	**1**	0	0	0	0	0
Versioning Debt	**1**	**1**	0	0	0	0	0
Usability Debt	**1**	0	0	0	0	0	0
Test Debt	4	**15**	14	5	3	3	2
Documentation Debt	7	2	**9**	6	4	1	1
Debt	0	1	**2**	0	0	1	0
Automation Test							
Requirement Debt	3	1	8	5	**12**	0	4

Caption:
#AD - Quantity of TD types found out within the category adopting of good practices for software development.
#CP - Quantity of TD types found out within the category controlling and measuring the quality in the project.
#FP - Quantity of TD types found out within the category following a well-defined project planning.
#HT - Quantity of TD types found out within the category having an effective team.
#CR - Quantity of TD types found out within the category controlling and negotiating the software requirements.
#ID - Quantity of TD types found out within the category identifying, managing, and estimating TD.
#FW - Quantity of TD types found out within the category following and improving a well-defined process.

category *following a well-defined project planning*, **HT** for the category *having an effective team*, **CR** for the category *controlling and negotiating the software requirements*, **ID** for the category *identifying, managing, and estimating TD*, and **FW** for the category *following and improving a well-defined process*. Thus, for example, **#AD** represents the quantity of TD types found out within the category *adopting of good practices for software development*.

Based on the high number (highlighted in bold and underlined in Table 7) of TD types contained in each category, we notice that *architecture debt*, *defect debt*, *design debt*, *infrastructure debt*, *process debt*, *service debt*, *usability debt*, and *versioning debt* items could be mainly prevented by considering preventive actions from the category *adopting of good practices for software development*. In addition to using this category, *code debt* and *design debt* items could be curbed by using preventive actions from the categories *having an effective team* and *identifying, managing, and estimating TD*, respectively. Preventive actions from the category *following a well-defined project planning* could be employed for preventing *test debt*, *documentation debt*, and *debt automation test* items. For *requirement debt* items, preventive actions from the category *controlling and negotiating the software requirements* seem to be a good starting point.

4.4 What are the impediments to preventing TD? (RQ4)

In the questionnaire (Q22), participants indicated whether it would be possible to prevent the TD item reported in Q13. Only 24 (~11%) of them answered that prevention could not be achieved, reporting eighteen situations that hinder the prevention of TD. The most cited reasons were *short deadlines* (7) (with 7 citations), *need to reduce time to market* (2), *lack of concern about maintainability* (2), *lack of technical knowledge* (2), and *lack of qualified professionals* (2). Despite the relatively small amount of response data available for this question, the results suggest that the inability to manage deadlines as well as issues involving the level of knowledge of the team make it difficult to implement preventive actions. The complete list of impediments is available at http://bit.ly/2LQFH6P.

5 IMPLICATIONS FOR PRACTITIONERS AND RESEARCHERS

Professionals who are initiating strategies to prevent the occurrence of TD in their projects can use two criteria for choosing preventive actions. The first criterion is to just consider the most commonly used preventive actions as a starting point. In this sense, the results reported in Fig. 1 and Table 3 would be useful since they provide a ranking of the actions and categories that have most commonly been considered in practice.

The second criterion is related to the combination of preventive actions considering TD types. By this criterion, if team has recurring problems with a specific TD type and is establishing a strategy to support TD prevention from scratch, it can identify the preventive actions that could be used together for a specific type of debt. Professionals may use the information presented in Tables 4-7 to this end. If the team already has some prevention actions in place and has recurring problems with a specified TD type, it could observe the tables to identify other actions that are strongly connected to the existing ones and consider including them in their current prevention strategy. For example, if team has problems with *code debt* items, according Table 7, preventive actions arising from the category *controlling and measuring the quality in the project* would be candidates for consideration.

Regarding the impediments to preventing TD, organizational actions that lead to better technical qualifications of team and negotiation of flexible deadlines with customers can also potentially lead to a favorable scenario that minimizes the occurrence of TD.

For researchers, the obtained results support the development of new research on actions and impediments to TD prevention. The presented top 10 list of preventive actions, the categories, the relationship between preventive actions, categories, and TD types, and the list of impediments can guide new investigations in a problem-driven way.

6 THREATS TO VALIDITY

As in any empirical study, there are threats to validity in this work. We attempted to remove them when possible and mitigate their

effect when removal was not possible. In this work, the primary threat to **conclusion validity** arises from the coding process as coding is mainly a creative task. To mitigate this threat, the coding process was performed individually by two researchers and reviewed by one experienced researcher.

Concerning the **internal validity**, the questionnaire represents the main threat that could affect this study. As indicated in [10], the questionnaire only has direct questions, avoiding misunderstanding that could lead to meaningless answers. Besides, the questionnaire has passed through successive validation tasks (three internal and one external) and a pilot study to detect any inconsistencies or misunderstandings before executing the survey.

Finally, we reduced the **external validity** threats by targeting industry professionals and seeking to achieve participant diversity among the survey respondents. However, although the population provides representative results on TD prevention, it came only from Brazilian and North American practitioners. Also, the population is mainly characterized by the presence of developers, while a more balanced distribution on the number of developers and projects managers could be interesting too. In search of more generalizable results, the *InsighTD* is now being replicated in Colombia, Costa Rica, Chile, and Finland.

7 CONCLUDING REMARKS

This work identifies actions being used in practice to prevent the occurrence of TD in software projects. It also identifies the main impediments that hamper the application of these actions. Further, we group the actions into categories and identify relationships among preventive actions and TD types, and among categories and TD types, indicating in both relationships how they can be combined to minimize the occurrence of TD types. Our contributions are significant because they provide early exploratory assessments of how industry practitioners deal with TD; which in turn can help focus and direct the research performed by academics.

The next steps of this research include: (i) to improve the external validity of the obtained results by aggregating data from other *InsighTD* replications, and (ii) to run deeper analyses to investigate whether TD preventive actions and impediments are impacted by variables such as used process model, participant experience and role, and organization/project size.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior – Brasil (CAPES) – Finance Code 001. 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. For more information on the award or Laureate, please visit www.laureate.net.

REFERENCES

[1] R.O. Spínola, A. Vetrò, N. Zazworka, C. Seaman and F. Shull. 2013. Investigating technical debt folklore: shedding some light on technical debt opinion. In *Proceeding of the 4th International Workshop on Managing Technical Debt* (MTD), San Francisco, CA, 1-7. DOI: 10.1109/MTD.2013.6608671.

[2] N. Zazworka, R.O. Spínola, A. Vetrò, F. Shull and C. Seaman. 2013. A case study on effectively identifying technical debt. In *Proceeding of the 17th Int. Conf. on Evaluation and Assessment in Software Engineering* (EASE). ACM, New York, NY, USA, 42-47. DOI: http://dx.doi.org/10.1145/2460999.2461005

[3] P Kruchten, RL Nord, and I Ozkaya. 2012. Technical debt: from metaphor to theory and practice. *IEEE Software*, Published by the IEEE Computer Society.

[4] A. Martini, J. Bosch, and M. Chaudron. 2014. Architecture technical debt: understanding causes and a qualitative model. In *Proceeding of 40th Euromicro Conf. on Software Engineering and Advanced Applications* (SEAA). IEEE Computer Society, Washington, DC, USA, 85-92.

[5] R. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas. 2012. In search of a Metric for Managing Architectural Technical Debt. In *Proceeding of Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture* (WICSA/ECSA), Helsinki, 91-100. DOI: 10.1109/WICSA-ECSA.212.17.

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

[7] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCoemack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan and N. Zazworka. 2010. Managing technical debt in software-reliant systems. In *Proceeding of the FSE/SDP workshop on Future of software engineering research* (FoSER). ACM, New York, NY, USA, 47-52. DOI=10.1145/1882362.1882373 http://doi.acm.org/10.1145/1882362.1882373

[8] J. Yli-Huumo, A. Maglyas, and K. Smolander. 2016. How do software development teams manage technical debt? - An empirical study. *Journal of System and Software*. 120, C (Oct. 2016), 195-218.

[9] Project Management Institute. 2017. *A Guide to the Project Management Body of Knowledge: PMBOK Guide* (6th. ed.), 14 Campus Boulevard Newtown Square, Pennsylvania, The United States.

[10] N. Rios, R.O. Spínola, M. Mendonça, and C. Seaman. 2018. The Most Common Causes and Effects of Technical Debt: First Results from a Global Family of Industrial Surveys. In *Proceeding of the 12th Int. Symposium on Empirical Software Engineering and Measurement (ESEM)*. ACM, New York, NY, USA, Article 39, 10 pages. DOI: https://doi.org/10.1145/3239235.3268917

[11] N. Rios, M. Mendonça, C. Seaman, and R.O. Spínola. 2019. Causes and Effects of the Presence of Technical Debt in Agile Software Projects. In *Proceedings of the 2019 Americas Conference on Information Systems* (AMCIS), Cancun, Article 3, 10 pages.

[12] N. Rios, R.O. Spínola, M. Mendonça, and C. Seaman. 2019. Supporting Analysis of Technical Debt Causes and Effects with Cross-Company Probabilistic Cause-Effect Diagrams. In *Proceeding of the 2nd International Conference on Technical Debt* (TechDebt), Montreal, QC, Canada, 3-12. DOI: 10.1109/TechDebt.2019.00009

[13] Carolyn Seaman and Yuepu Guo. 2011. *Measuring and monitoring technical debt*. 1. ed. [s.l.] Elsevier Inc., 2011. v. 82.

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

[15] N. Rios, R.O. Spínola, M. Mendonca, and C. Seaman. 2018. A Study of Factors that Lead Development Teams to Incur Technical Debt in Software Projects. In *Proceeding of the 44th Euromicro Conference on Software Engineering and Advanced Applications* (SEAA), Prague, 429-436. DOI: 10.1109/SEAA.2018.00076

[16] Carolyn Seaman. 1999. Qualitative methods in empirical studies of software engineering. *IEEE Trans. on Soft. Engineering*, 25(4):557-572.

[17] Anselm Strauss and Juliet M. Corbin. 1998. *Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory*. Sage Publications.

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