Causes and Effects of the Presence of Technical Debt in Agile Software Projects

Completed Research

Nicolli Rios
Department of Computer Science
Federal University of Bahia
Salvador, BA, Brazil
nicollirioss@gmail.com

Manoel Mendonça
Dept. of Computer Science, Federal Univ. of Bahia and Fraunhofer Project Center
Salvador, BA, Brazil
manoel.mendonca@ufba.br

Carolyn Seaman
Department of Information Systems
University of Maryland Baltimore County
Baltimore, MD, United States
cseaman@umbc.edu

Rodrigo Oliveira Spínola
Graduate Program in Systems and Computing, Salvador University and State University of Bahia
Salvador, BA, Brazil
rodrigo.spinola@unifacs.br

Abstract

The current software development scenario is characterized by a wide adoption of agile methodologies. Despite its benefits, agile software development (ASD) is also vulnerable to technical debt (TD). In fact, due to its delivery-oriented nature, ASD seems to be more prone to TD accumulation than traditional software development. Although the concept of TD has been increasingly investigated in software engineering, there is a lack of studies regarding TD in ASD. This paper discusses the most common causes and effects of TD in ASD. Through an industrial survey on TD, we compiled the answers of 51 practitioners to identify 43 causes and 30 effects of TD in ASD. This information can support decision-makers on how to deal with TD in agile software development.

Keywords

Technical debt, technical debt causes, technical debt effects, agile software project, survey.

Introduction

Technical debt (TD) contextualizes the problem of outstanding software development tasks (for example, tests planned but not executed, pending documentation update, use of bad design practices, code that does not exhibit good coding practices) as a kind of debt that brings a short-term benefit to the project (normally in terms of higher productivity or shorter release time of software versions), that may have to be paid with interest later in the development process (for example, a poorly designed class tends to be more difficult and costly to maintain than if it had been implemented good object-oriented practices) (Kruchten *et al.*, 2012; Guo *et al.*, 2014). It is common for a software project to incur in TD during the development process, because small quantities of debt can increase productivity. However, the presence of the debt creates risks to the project, making its management more difficult (Kruchten *et al.*, 2012; Rios *et al.*, 2018c).

The current software development landscape is characterized by wide adoption of agile methodologies, which define processes and introduce practices that address a range of problems currently faced by development teams (Martin, 2003). Despite its expected benefits, agile software development (ASD) is also vulnerable to the presence of technical debt (Holvitie *et al.,* 2018). This issue deserves investigation, because agile software development seems to be more prone to TD accumulation than traditional software development approaches, due to its delivery-oriented nature (Kurapati *et al.*, 2012). This characteristic can lead, for example, development teams to compromise the internal quality of a software by designing an architecture that is far from ideal. As consequence, organizations that apply agile methodologies in their projects may

eventually need more time to release versions, have increased costs and problems to maintain the software quality.

Although the concept of technical debt has been increasingly investigated in the software engineering field over the past years (Rios *et al.*, 2018b), there still is little investigation on how ASD can accommodate the concept of technical debt. To shed some light in this discussion, Holvitie *et al.* (2018) and Caires *et al.* (2018) conducted a survey with practitioners from Finland, Brazil and New Zealand on how technical debt issues relate to agile software processes and practices. They investigated the participants' level of knowledge on technical debt, how technical debt manifests itself in their projects, and what processes and practices of ASD are sensitive to it. In general, the study pointed out that the processes and practices that are closest to the implementation and maintenance activities are perceived as having the most positive effects on the control of technical debt. On the other hand, the authors identified that TD items usually come from problems in the software architecture.

In another work, Santos *et al.* (2013) proposed the use of a TD board in ASD. Their goals were to measure, manage and visualize the main technical debt categories in ASD. According to them, the proposed approach improved the teams' TD awareness, enhanced communication regarding technical decisions, and stimulated a beneficial competition between teams towards debt payment.

Despite the contributions of these works, there is a lack of knowledge on the relation between technical debt and agile software development. The area needs to better understand the factors that lead development teams to incur into technical debt and, also, the effects of the presence of debt in agile software development. Studying the underlying causes of technical debt, thus helping to identify actions that could prevent the TD items in the first place, is not yet a common practice. Neither is the careful examination of the effects of technical debt, which could then support prioritization strategies for paying off debt (Yli-Huumo *et al.*, 2014; Alves *et al.*, 2016; Rios *et al.*, 2018b). Setting up technical debt prevention practices is especially helpful to inexperienced developers (Yli-Huumo *et al.*, 2014). From the other perspective, understanding the TD effects can aid the prioritization of TD payment, and support more precise impact analysis and the definition of corrective actions to minimize possible negative consequences of TD to a project. In this context, this paper presents the top 10 most common causes and effects of technical debt in agile software development.

To identify common causes and effects of technical debt, we are running a globally distributed family of industrial surveys on technical debt, InsighTD. InsighTD is designed to run as an incremental large-scale study based on continuous and independent replications of the questionnaire in different countries. It currently involves researchers from eleven countries (Brazil, Chile, Colombia, Costa Rica, Finland, Italy, Norway, the Netherlands, Saudi Arabia, Serbia, and the United States).

The design of InsighTD, as well as the initial results of its execution in Brazil, was previously presented by Rios *et al.* (2018c). Different from this previous paper, in which we discussed the causes and effects of technical debt regardless of the process model used by the software development teams, this paper focuses on causes and effects of TD in agile software development.

In total, 51 practitioners from the Brazilian software industry answered the questionnaire considering their experience with technical debt in agile software development. The results allowed us to identify 43 causes and 30 effects of technical debt in ASD. Among the causes, we have identified issues related to project planning, technical knowledge, and people issues, such as *short deadlines, inappropriate planning*, *lack of knowledge*, *team overload*, and *lack of commitment*. Among the effects, we identified issues such as *low maintainability, delivery delay, rework*, and *financial loss*. Together, this knowledge on causes and effects of technical debt in ASD may provide useful information to support decision-makers on how to deal with technical debt, a set of information that is still missing in the technical literature.

Besides this introduction, this paper contains six other sections. The Background section presents an overview on technical debt and its relation to agile software development. The InsighTD section briefly presents the design of InsighTD, the survey's questions and data analysis procedures considered in this work. The Results section presents the findings of the study. Next section discusses the obtained results. We then discuss threats to the validity of the performed work and how they were mitigated. Finally, the last section contains final remarks and next steps of the on-going research.

Background

This section overviews relevant background on technical debt and review past work on agile software development from a technical debt perspective.

Technical Debt

The concept of technical debt was first mentioned by Cunningham (1992). The concept contextualizes problems faced during the software development considering tasks that are not carried out adequately. Software technical debt is a type of debt that brings a short-term benefit (e.g. increased development speed or shortened time to market), but which may have to be paid with interest later on the software development life cycle (Kruchten *et al.*, 2012; Guo *et al.*, 2014). Non-execution of tests, pending code refactoring, and outdated documentation are examples of TD. Currently, TD is recognized as a critical issue in the software development industry and is considered detrimental to software developing companies (Besker *et al.*, 2017). It can result in unexpectedly large cost overruns, severe quality issues, and the inability to add new features to the delivered software (Rios *et al.*, 2018c).

Some studies have investigated causes and effects of technical debt in software projects. In 2014, Yli-Huumo *et al.* carried out a study with the objective to understand if TD really exists, what its origin and the effects of its presence in software development. To achieve this objective, they carried out an exploratory case study of two independent software product lines. The results revealed several situations of technical debt in the studied product lines. The authors classified the TD as intentional or unintentional debt. The primary cause of intentional debt was the lack of development time, followed by pressure on the team. The most common causes of unintentional debt were lack of coding standards, programmer inexperience, lack of documentation, and lack of knowledge about future changes.

Martini *et al.* (2014) conducted a case study to investigate factors that cause architectural TD accumulation. Some causes identified by the study were business factors, pressure, functional prioritization, incomplete refactoring, and lack of documentation. In the end, the authors presents a taxonomy of the causes of architectural technical debt and a model representing the trends in relation to the accumulation and payment of architectural technical debt over time.

Ernst *et al.* (2015) conducted a survey of software engineers and architects working on long-term projects. The purpose of the study was to understand the relationship between engineers and TD, and whether tools and techniques are used for their management. The study identified that decisions about design architecture are the most important cause of technical debt. In addition, participants indicated that the existing management tools are not suitable to avert technical debt.

Recently, Rios *et al.* (2018a) investigated causes that lead to the occurrence of technical debt, if these causes occur in isolation or in combination, if technical debt can be prevented, and, in terms of effort, if it is better to prevent debt, or incur in it and pay it off later. Through an interview-based case study with ten practitioners from two software organizations, the authors identified 57 causes that lead a development team to incur in debt. The result indicated that debt can be prevented, and it is better to work on prevention activities than to pay off debt later. For most technical debt types, TD causes occurred in combination.

Finally, Rios *et al.* (2018c) discussed the basic survey design and the preliminary results of the first round of InsighTD execution in Brazil. InsighTD is an initiative which aims at establishing an open and generalizable set of empirical data on practical problems of technical debt. Its initial study identified 79 causes and 66 effects of technical debt. Our work complements this study by analyzing the data under the perspective of agile software development.

Agile Software Development and Technical Debt

Some studies have investigated the relation between technical debt and agile software development. Codabux and Williams (2013) performed an industrial case study to determine the best practices and known challenges with agile adoption and the management of technical debt. The results showed that the developers considered their own technical debt taxonomy based on the type of work assigned to them and their personal understanding of the term. Despite high-level management categories, developers have considered design, test, and defect debt most of the time. In addition to developers having their own taxonomy, assigning

dedicated teams for technical debt reduction and allowing other teams about 20% of the time per sprint for debt reduction are good initiatives towards lowering technical debt.

In another work of the area, Soares *et al.* (2015) investigated the difficulties when working with agile requirements and if those difficulties create a favorable scenario for incurring documentation debt in agile software projects. Based on the results of a controlled literature review followed by an exploratory study, the authors presented a list of factors that may lead development teams to incur documentation debt when working with agile requirements. In 2016, Mendes *et al.* carried out a retrospective study on a software project that was developed using user stories. The study aimed to investigate the impact of documentation debt to agile software projects. The results indicated an extra effort of ~47% of the total effort estimated for developing the project. According to the authors, documentation debt can cause significant impacts in terms of maintenance effort and cost on software projects that use agile requirements in their development.

Behutiye *et al.* (2017) performed a systematic literature review to analyze and synthesize the state of the art of technical debt in the context of agile software development. The results indicated five research areas of interest of TD in agile software development. Among them, TD management received the highest attention, followed by architectural debt. The authors also found 12 strategies for the management of TD in agile software development, of which *refactoring* and *enhancing the visibility of TD* were the most significant.

More recently, Holvitie *et al.* (2018) carried out a survey to classify the effects of the adoption of agile methodologies in TD management. The study indicated that practitioners are aware of technical debt, and that it is usually found in legacy systems. In addition, the results indicated that the analyzed agile practices and processes contribute to the reduction of technical debt In particular, techniques that verify and maintain the structure and clarity of the implemented artifacts positively affect the TD management, like coding and refactoring standards. Complementing the work of Holvitie *et al.* (2018), Caires *et al.* (2018) investigated the effects of agile practices and process from the perspective of the Brazilian software industry. The results indicated that many instances of technical debt reside in software implementation and occur due to deficiencies in its architecture. In addition, the authors also pointed out that the size of a debt item is proportional to its impact on the project. Finally, most of the respondents indicated that *refactoring* and *iteration* are the agile practice and process that have the most positive effect on technical debt.

Agile software development is highly vulnerable to the occurrence of technical debt due to its characteristic of being focused on fast delivery. While TD has become a well-regarded concept in the agile community, there is still limited initial evidence in the technical literature relating both areas. The size of the samples considered in the studies tends to be small and most studies consider only documentation or architecture debt. This work contributes to the area by discussing the causes and effects of technical debt for agile software projects. The research project in which the causes and effects are grounded is discussed in the next section.

The InsighTD Project

InsighTD is a globally distributed family of industrial surveys, planned cooperatively with technical debt researchers from different research centers around the world (Rios *et al.*, 2018c). The project aims to organize an open and generalizable set of empirical data on the causes and effects of technical debt in software projects. Its design establishes the foundations for the survey to be continuously replicated in different countries. The idea behind these replications is to pursue generalizable results about the state of practice in the TD area. The rationale behind choosing countries as scopes of replication is twofold: (i) organizing the work and making the dissemination of the survey wider, and (ii) investigating whether differences in local development practices could influence how participants experience the TD concept.

Currently, researchers from 11 countries (Brazil, Finland, Netherlands, Italy, Norway, Saudi Arabia, Serbia, United States, Colombia, Chile, and Costa Rica) have already joined the project. The execution of InsighTD in Brazil has already concluded. The North American, Colombian, Costa Rican, and Serbian replication teams have just begun their replications.

In this section, we summarize how we asked participants about the causes and effects of technical debt and discuss the data analysis procedures. Further information on the design of InsighTD and other initial results of the research can be found in (Rios *et al.*, 2018c).

Questions on Causes and Effects of Technical Debt

The InsighTD questionnaire consists of 28 questions. Of these, four are related to causes and two to effects of technical debt. We first asked participants to describe a particular real example of technical debt (Q13), then asked them to describe the cause that led to that example (Q16), what other cause or factor contributed to that first mentioned cause (Q17), and what other motives, reasons or causes contributed directly or indirectly to the example of TD item (Q18). Next, we asked for an ordered list of five causes that most contribute to technical debt occurrence over the participant's entire experience (Q19). We also asked what effects were felt in the project of the given example (Q20) and, then, asked for an ordered list of five effects that have the greatest impact on a project, again considering the participant's entire experience (Q21). The other questions of the questionnaire intends to characterize the participants and, also, collect some data that helps to understand how TD has been managed in practice. In this work, we will consider only questions Q13 to Q21 and the characterization questions.

To analyze these responses, we applied qualitative data analysis techniques (Strauss and Corbin, 1998; Seaman, 1999). As we had no prior expectation about the content of the responses, we followed an inductive logic approach to generate a new theory based on the given qualitative data. We applied manual coding as follows. Initially, the first and second authors individually coded the set of all answers for two subsets (causes and effects) of related questions. This involves open coding as described in (Strauss and Corbin, 1998), then axial coding to derive higher level categories. Next, they discussed possible differences in their coding until they reached consensus. Thus, we analyzed the data by attaching codes to small coherent units in the answers and categorizing the emerged concepts (causes/effects) in a hierarchy of categories. This process was performed iteratively until reaching a state of saturation.

Besides these questions, we also define some specific questions to characterize the survey participants and their respective organizations.

Results

The first execution of the InsighTD survey was conducted in Brazil from December 7, 2017, to January 7, 2018. Invitations were sent by e-mail or LinkedIn profile. At LinkedIn, we searched by software professionals who work (or worked) in several areas of software development (e.g., testing, coding, documentation, requirements, and management). In total, we sent the survey invitation to about 513 professionals and 112 of them completed the questionnaire. Fifty-one professionals indicated that were working with agile software development. This is the set of answers that we considered for analysis in this paper.

Demographics Data

Several types of expertise were represented in the participant sample, as we can observe in Table 1. Participants defined their level of experience in their role among the following options: Novice (Minimal or "textbook" knowledge without connecting it to practice), Beginner (Working knowledge of key aspects of practice), Competent (Good working and background knowledge of area of practice), Proficient (Depth of understanding of discipline and area of practice), and Expert (Authoritative knowledge of discipline and deep tacit understanding across area of practice). Most of them are proficient (35%) in their role, followed by competent (33%), expert (20%) and beginners (12%), indicating that, in general, the questionnaire was answered by professionals with experience in their functions.

Role	*#*	*%*	*Novice*	*Beginner*	*Competent*	*Proficient*	*Expert*
Developer	20	38%	0	1	9	6	4
Software Architect	8	16%	0	0	0	6	2
Project Leader /Project Manager	8	16%	0	1	3	3	1
Test Manager / Tester	6	12%	0	0	3	2	1
Requirements Analyst	3	6%	0	2	1	0	0
Process Analyst	2	4%	0	1	0	1	0
Infrastructure analyst	1	2%	0	0	1	0	0
Business Analyst	1	2%	0	1	0	0	0
Performs multiple functions	1	2%	0	0	0	0	1
Configuration Manager	1	2%	0	0	0	0	1

Table 1. Participants' Roles

Organizations of different sizes are represented in the dataset. They are well distributed among small (31%), mid (47%), and large size (22%) companies. By observing the data in more detail, we can see that most participants (mode) work in organizations with more than 51-250 employees, closely followed by enterprises with 11-50 employees. Therefore, the participants tend to work in mid companies, but we have representatives from companies of all sizes.

Concerning the size of the project teams, most of them consisted of 10-20 people (31%), although 29% had fewer than five staff. In addition, a significant number (25%) indicated a team of 5-9 people. The median is teams composed of 5-9 people, indicating that participants tend to work in small development teams, but we have representatives from teams of all sizes.

The most common system age was 1 to 2 years old (35%), closely followed by 2-5 years old (31%). We also had a significant number of represented systems of less than 1-year-old (21%), 5-10 years old (10%), and only one project with more than 10 years. Finally, the systems were typically between 10 KLOC and 1 million SLOC in size (62%). But we also had samples for smaller (<10 KLOC – 21%) and larger systems (>10 MLOC – 9%).

Thus, overall, the collected data seems to be a good representation of the Brazilian software industry diversity, reaching (i) several participants' roles and levels of experience, (ii) organizations of different sizes, and (iii) projects of different age, size, and team size.

Causes of Technical Debt in Agile Software Projects

Overall, 43 causes were identified in this study. Two researchers analyzed each participant's answers to Q16-Q19, identifying the causes by directly copying the terms provided in the answers. To standardize the nomenclature, small adjustments were made without altering the semantics of the labels (for example, deadline and short deadline were mapped to deadline). The causes were then filtered to exclude duplicates. If a participant cites the same cause for Q16-Q18 and Q19, then, the overall frequency of that cause is incremented only by one.

The top 10 most cited technical debt causes on agile projects, as informed by the 51 participants in Q16-18, are visualized together with the frequency in which they appear in the list of the causes most likely to lead to technical debt (reported in Q19) in Figure 1. We can observe that *deadline* is the most cited cause in general and as a most likely factor, indicating that it is a factor that normally contributes to the occurrence of debt items. *Non-adoption of good practices, inappropriate planning* and *lack of knowledge* are other causes cited by at least 14% of the participants. Also were cited by at least 8% of the participants, causes such as *bad design, lack of knowledge of technology, required infrastructure unavailable* and *team overload*.

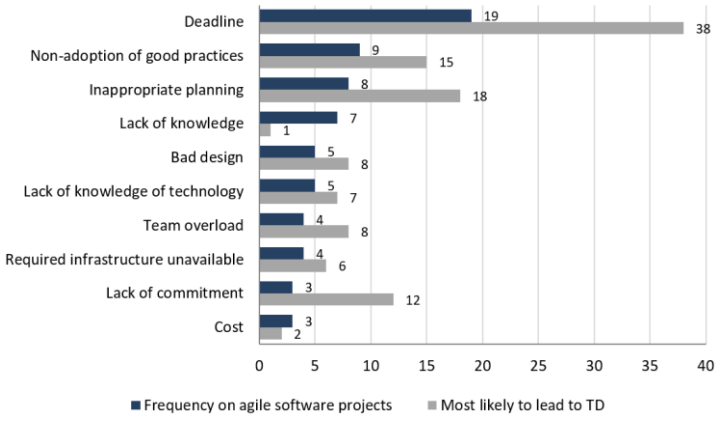

Figure 1. Top 10 technical debt causes cited in agile projects

By analyzing Figure 1, we can also observe that for some causes, even if they seem not to occur as often as others, they seem to be still more likely to lead to the occurrence of technical debt. For instance, lack of commitment is the ninth most frequently cited cause in the examples but it ranks in the fourth position among causes most likely to lead to technical debt in general. Therefore, it might be considered more important than, for example, lack of knowledge, which was mentioned more frequently in examples but less frequently in lists of causes most likely to lead to technical debt. The results also indicate that technical

reasons do not seem to be the most commonly remembered causes of technical debt. Of the 10 causes cited for agile projects, only four of them (*non-adoption of good practices*, *lack of knowledge of technology*, *bad design*, and *required infrastructure unavailable*) are directly linked to technical issues. Thus, non-technical issues seem to have a significant role in the occurrence of TD items in agile software development. The team's expertise and maturity are decisive for the occurrence of the technical debt, as demonstrated by the presence of the following cited causes: *lack of knowledge* and *lack of commitment*. In addition, planning issues (*deadline, inappropriate planning, team overload,* and *cost*) were also frequently cited.

Effects of Technical Debt in Agile Software Projects

We identified a total of 30 effects of technical debt for agile software projects. To reach this result, the first and second authors of this work analyzed each participant's responses to Q20 and Q21, identifying the effects through direct copying of the terms described by participants. When necessary, to standardize the nomenclature, small adjustments were made without changing their semantics (for example, *difficulty understanding the code* and *reduced maintainability* were mapped to *low maintainability*). The effects were then filtered to exclude duplicates.

The top 10 most cited effects (based on data from Q20) together with the frequency in which they appear in the list of the effects that have a bigger impact (Q21) is presented in Figure 2. As we can observe, three effects stand out: *low maintainability, delivery delay,* and *rework*, occupying the first three positions in the rank of the most cited effects. *Low maintainability* encompasses problems that occur during software maintenance activities, such as increased effort to fix bugs as well as limitation in system evolution. *Delivery delay* refers to the non-fulfillment of the deadlines agreed with the customer. *Rework* refers to the necessity of redoing an activity during the development process because it was not properly done in the first time.

Another effect that was commonly cited by participants as having a bigger impact was *low quality*, that refers to any aspect that reduces the quality of an artifact (including errors and known defects that are not fixed). This finding suggests that despite the participants have not faced this effect too much frequently in their projects (~8%), development teams need to be watchful about it.

Finally, we can also observe in Figure 2 that there are two effects related to relations among people: *team demotivation* and *stress with stakeholders*. This is a signal that the presence of technical debt can undermine the working environment in agile software development. The complete list of causes and effects of technical debt can be accessed at https://goo.gl/PttnN8.

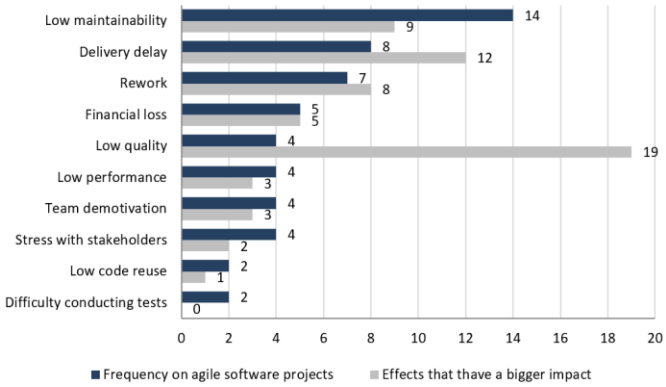

Figure 2. Top 10 technical debt effects cited in agile projects

Discussion

Causes and Effects of Technical Debt

The high number of causes and effects of technical debt found in this study can make it difficult to make practical use of this information. Thus, we built probabilistic cause-effect diagrams to provide a more structured view of the identified causes and effects. Probabilistic cause-effect diagrams can provide visual support in causal analysis sessions by representing knowledge about the causes and common effects of problems collected from previous experiences (Rios *et al.*, 2019). Such diagram allows development teams to answer questions such as: "Given the experience on previous projects, how likely a given cause is leading to a specific problem?" or "Given the experience on previous projects, how likely is a given effect is going to occur as a consequence of a specific problem?".

Figure 3 shows the elaborated cause-effect diagrams for the causes and effects of technical debt in agile software development. To create the diagrams, we considered the causes and effects of technical debt cited by participants in questions Q16-Q18 and Q20. In the left side, the closer a cause is to the main line, the greater the probability that it will lead to the occurrence of technical debt in the project. For example, the diagram indicates that the causes that lead to technical debt are usually related to planning and management issues (~34% of the time) in agile software development. Within this category, the most commonly cited cause is *deadline*, followed by *inappropriate planning*. In the right side, the closer the effect is to the main line, the more likely it will be felt in the project. For example, the diagram indicates that quality issues (~28% of the time) are the most common category of effects felt by development teams. Within this category, the most commonly cited effect is *rework*, followed by *low quality* and *low performance*.

It is noteworthy to mention that the causes and effects indicated in the diagram will not necessarily be those that impact a particular agile software project. However, the diagram is still a valuable source of information. The represented set of information can lead development teams to think "out of the box" and visualize possible causes or effects that they would not have thought of if they did not have access to the diagram, which represents the state of practice in terms of causes and effects of TD in agile software development.

Implications for Practitioners and Researchers

Together, knowledge about the causes and effects of TD provides useful information to support decision-makers on how to deal with TD in agile software development. This set of information fills a gap in the technical literature. Moreover, as discussed in the previous section, the elaborated probabilistic cause-effect diagrams can provide a broad and practical view over the collected data on the causes and effects of TD.

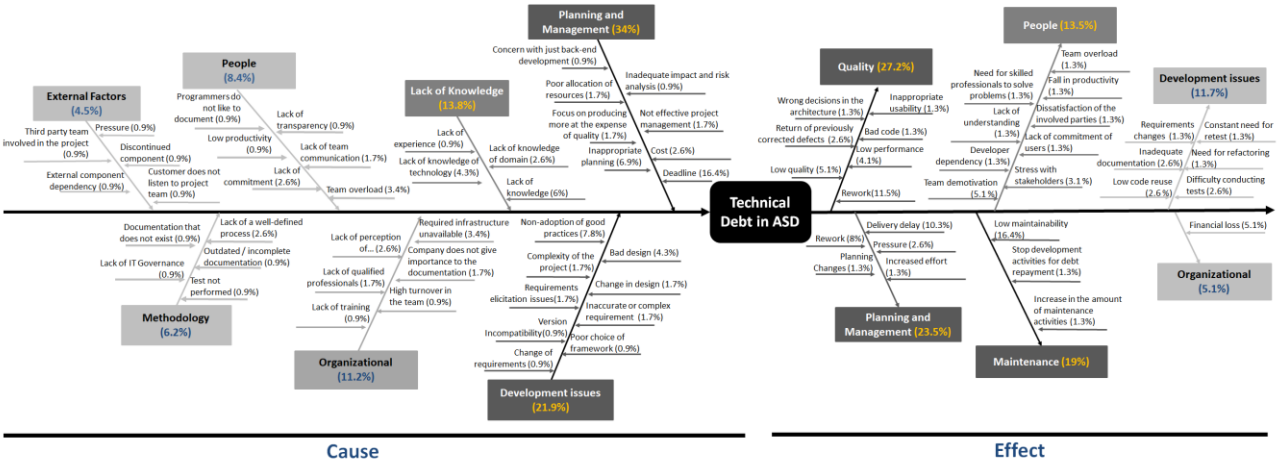

Figure 3. Probabilistic cause-effect diagram for the causes and effects of TD in agile projects

Development teams can use the list of causes to understand factors that contribute to the occurrence of technical debt and, if necessary, work on preventive actions. For example, if a team recognizes such common causes as *lack of knowledge, lack of knowledge of technology, non-adoption of good practices, or team overload* as relevant to their own context, they could use avoidance of TD as further motivation to improve the workforce through hiring or training practices. The value of alleviating some of the non-technical causes, such as inappropriate planning or team overload, is enhanced by viewing these factors as causes of technical debt. In addition, considering the list of identified effects, practitioners can have a clearer view of possible impactful consequences of technical debt in their projects. This could aid decision making, by incorporating a wider range of potential consequences into decision models that try to capture the long-term cost of TD.

For researchers, our results support future research by providing insights into the perspectives of practitioners about the causes and effects of technical debt. The presented lists allow researchers to guide their research in a problem-driven way.

Threats to Validity

As in any empirical study, we have threats to validity in this work. We attempt to remove them when possible and mitigate their effect when is not possible to remove them.

Construct validity: some social threats to construct validity related to the behavior of participants and researchers may arise in this study. Participants may act differently based on the fact they are participating in a study (Wholin *et al.*, 2012). To help avoid hypotheses guessing and evaluation apprehension (Wholin *et al.*, 2012), in the invitation email, we clearly explain the purpose of the study and ask participants to answer questions based on their own experience. We also inform that the questionnaire and the data collection do not take into consideration the identity of the respondents. Regarding the researcher's expectations, in which experimenters can bias the results of a study based on what they expect from the experiment, we involved different researchers in internal and external reviews of the questionnaire. Thus, this threat was minimized.

Conclusion validity: in this work, the greatest threat to the validity of conclusion stems from the coding process of the InsighTD data, since coding is essentially a creative task. To mitigate this threat, we first conducted a pilot study in the analysis. After agreeing with the first resulting codes, like common understanding about the text and the abstraction level in the codes, the coding process was done individually by two researchers. They then discussed the results until they reached consensus.

Internal validity: maturation and instrumentation are two threats to internal validity that may affect this study. If instrumentation is badly designed, the study results are negatively affected (Wholin *et al.*, 2012). To deal with this threat, the questionnaire was designed in a way that we have only direct questions and, thus, requiring as little interpretation as possible, avoiding a misunderstanding that would lead to meaningless answers. In addition, the questionnaire has passed through several validation steps (three internal and one external) and a pilot to avoid inconsistencies or misunderstanding before running the survey.

Maturation indicates that participants may react differently as time passes, for example, if the questionnaire is too long (Wholin *et al.*, 2012). In our study, we tried to minimize this threat by avoiding a questionnaire that takes too much time to be answered. During the pilot study, we found that the mean time to complete the full questionnaire was about 20 minutes. Another sign that this threat was not raised was that all participants answered the questionnaire in full.

External validity: threats to external validity are conditions that limit our ability to generalize the results (Wholin *et al.*, 2012). We reduce this threat by achieving diversity of participants who responded the survey. Besides, in search of more generalizable results, InsighTD is based on continuous replications of the questionnaire worldwide.

Final Remarks

This paper reports the initial results of a global family of industrial surveys, InsighTD, considering the point of view of 51 practitioners from the Brazilian software industry that work with agile software development. The main contribution of this work is the list of causes and effects of technical debt that represents the reasons that lead software teams to incur debt and the pain that developers suffer because of its presence in agile software projects. This set of information fills a gap in the technical literature.

The next steps of this research include: (i) continuously synthesize of data on causes and effects of technical debt in agile software development based on the replications of InsighTD worldwide; (ii) investigate differences and similarities of causes and effects of technical debt considering the process model (agile or traditional) as parameter of comparison; (iii) investigate what types of debt have been most commonly faced by software engineers in agile software development, and; (iv) running other possible analyses that can be performed based on the characterization of participants. Replications of InsighTD are currently being performed in the United States, Colombia, Costa Rica, and Serbia. With the results of these replications, we expect to achieve further findings about the relation between agile software development and technical debt.

Acknowledgements

This work was supported by the Coordination for the Improvement of Higher Education Personnel - Brazil (Capes), under the Capes/IIASA Sandwich Doctoral Program, process nº 88881.189667 / 2018-01.

REFERENCES

Alves, N., Mendes, T., Mendonça, M., Spínola, R., Shull, F., Seaman, C. 2016. Identification and Management of Technical Debt: A Systematic Mapping Study. Information and Software Technology, 70, 100 – 121.

Behutiye, W. N., Rodríguez, P., Oivo, M., and Tosun, A. 2017. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. Information and Software Technology, 82, 139-158.

Besker, T., Martini, A., and Bosch, J.. 2017. Time to Pay Up Technical Debt from a Software Quality Perspective. In Proc. of the 20th Ibero-American Conference on Software Engineering. Argentina.

Caires, V., Rios, N., Holvitie, J., Leppanen, V., Mendonca, M., Spínola, R. O. 2018. Investigating the Effects of Agile Practices and Processes on Technical Debt - The Viewpoint of the Brazilian Software Industry. In: The 30th Int. Conf. on Software Engineering & Knowledge Engineering, San Francisco.

Codabux, Z. and Williams, B. 2013. Managing technical debt: An industrial case study. In Proceedings of the 4th International Workshop on Managing Technical Debt (pp. 8-15). IEEE Press.

Cunnigham, W. 1992, The WyCash Portfolio Management System, in Addendum to the proceedings on Object-oriented programming systems, languages, and applications, pp. 29-30.

Ernst, N.A., Bellomo, S., Ozkaya, I., Nord, R.L., and Gorton, I. 2015. Measure it? Manage it? Ignore it? software practitioners and technical debt. In Proc. of the ESEC/FSE 2015, pp. 50-60.

Guo, Y., Spínola, R.O., and Seaman, C. 2014. Exploring the costs of technical debt management – A case study. Empirical Soft. Engineering, v. 1, p. 159-182. DOI=http://dx.doi.org/10.1007/s10664-014-9351-7.

Holvitie, J., Licorish, S. A., Spínola, R. O., Hyrynsalmi, S., Macdonell, S. G., Mendes, T. S., Buchan, J., Leppänen, V. 2018. Technical debt and agile software development practices and processes: An industry practitioner survey. Information and Software Technology, v. 96, p. 141-160.

Kruchten, P., Nord, R., Ozkaya, I. 2012. Technical Debt: From Metaphor to Theory and Practice, in IEEE Software, vol. 29, no. 6, pp. 18-21, Nov.-Dec. DOI: 10.1109/MS.2012.167

Kurapati, N., Manyam, V. S. C., and Petersen, K. 2012. Agile software development practice adoption survey, in: Agile processes in software engineering and extreme programming, Springer, pp. 16-30.

Martin, R.C. 2003. Agile software development: principles, patterns, and practices. Prentice Hall.

Martini, A., Bosch, J., and Chaudron, M. 2014. Architecture Technical Debt: Understanding Causes and a Qualitative Model. In Proceedings of the 40th Euromicro SEAA, p. 85-92.

Mendes, T. S., Farias, M.A., Mendonça, M., Soares, H.F., Kalinowski, M., and Spínola, R.O. 2016. Impacts of agile requirements documentation debt on software projects: a retrospective study. In Proceedings of the 31st Annual ACM Symposium on Applied Computing (pp. 1290-1295).

Rios, N., Spínola, R. O., Mendonca, M., and Seaman, C. 2018a. A Study of Factors that Lead Development Teams to Incur Technical Debt in Software Projects. In: The 44th Euromicro Conference on Software Engineering and Advanced Applications, Praga.

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

Rios, N., Spínola, R. O., Mendonca, M., and Seaman, C. 2018c. The most common causes and effects of technical debt: first results from a global family of industrial surveys. In Proceedings of the 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '18). ACM, New York, NY, USA, Article 39, 10 pages. DOI: https://doi.org/10.1145/3239235.3268917

Rios, N., Spínola, R. O., Mendonca, M., and Seaman, C. 2019. Supporting Analysis of Technical Debt Causes and Effects with Cross-Company Probabilistic Cause-Effect Diagrams. In the Proceedings of the 2nd International Conference on Technical Debt (TechDebt 2019), Montreal.

Santos, P.S.M., Varella, A., Dantas, C.R., Borges, and D.B. 2013. Visualizing and Managing Technical Debt in Agile Development: An Experience Report. Lecture Notes in Business Information Processing. p 121-134.

Seaman, C. 1999. Qualitative methods in empirical studies of software engineering. IEEE Transactions on Software Engineering, 25(4):557-572.

Soares, H. F., Alves, N. S., Mendes, T. S., Mendonça, M., and Spínola, R.O. 2015. Investigating the link between user stories and documentation debt on software projects. In the 12th International Conference on Information Technology-New Generations (ITNG), pp. 385-390.

Strauss, A. and Corbin, J.M. 1998. Basics of Qualitative Research: Techniques and Procedures for Developing Grounded Theory. Sage Publications, Thousand Oaks.

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., and Wesslén, A. 2012. Experimentation in Software Engineering: An Introduction. Springer

Yli-Huumo, J., Maglyas, A., and Smolander, K. 2014. The sources and approaches to management of technical debt: a case study of two product lines in a middle-size Finnish software company. In: PROFES 2014. LNCS, vol. 8892, pp. 93–107. Springer. doi:10. 1007/978-3-319-13835-0_7