Department: Head
Editor: Name, xxxx@email

Technical and Non-Technical Prioritization Schema for Technical Debt: Voice of TD-Experienced Practitioners

Vladimir Mandić
University of Novi Sad, Serbia

Nebojša Taušan
INFORA Research Group, Serbia

Robert Ramač
University of Novi Sad, Serbia

Sávio Freire
Federal University of Bahia, Brazil

Nicolli Rios
Federal University of Rio de Janeiro, Brazil

Boris Pérez
Universidad de Los Andes, Colombia

Camilo Castellanos
Universidad de Los Andes, Colombia

Darío Correal
Universidad de Los Andes, Colombia

Alexia Pacheco
University of Costa Rica, Costa Rica

Gustavo Lopez
University of Costa Rica, Costa Rica

Clemente Izurieta
Montana State University, Montana, USA

Davide Falessi
University of Rome "Tor Vergata," Italy

Carolyn Seaman
University of Maryland Baltimore County, Maryland, USA

Rodrigo Spínola
Salvador University, Brazil

***Abstract*—Technical Debt (TD) can be injected at any stage of software development. Once injected, TD rarely remains contained and spreads across other stages causing various problems. During software development, technical and non-technical roles need to cooperate and communicate complex issues to implement optimal solutions. This article presents a model for conceptualizing TD causes, effects, payment practices, and payment avoidance reasons with a prioritization schema for technical and non-technical roles. The model is based on the analysis of 168 responses from TD-experienced practitioners and aims to support: (a) communication—by highlighting the similarities and differences between two perspectives, and (b) complementing the existing TD management and prioritization approaches with a conceptual model that takes into consideration the contextual factors and presents them in a form of prioritized lists of the most significant factors.**

IN THE LAST DECADE, technical debt (TD) research advanced from the notion of a metaphor toward specific engineering practices and tools for identifying, monitoring, and remedying TD issues. Many of these advances are related to the most obvious software artifact—*source code*. However, nowadays we know that TD can be injected into almost any artifact and during any stage of software development leading to different flavors of technical debt—*TD types* [1], [2]—e.g., code, architecture, design, documentation, test, process, or people debt. This is understandable since software development is a collaborative effort of practitioners working on and deciding about a number of issues, which may be closer to the source code artifact, thus more *technical*, or more distant from it, i.e., *non-technical*. All those decisions impact the creation of the product and consequently may lead to the creation of technical debt [2].

The importance of the technical and non-technical perspectives, although apparent, is still not fully comprehended when it comes to reasoning about TD. In a way, it is easy to imagine situations when due to the misalignment of technical and non-technical priorities, sub-optimal decisions are made, resulting with the injection of new TD. After all, the introduction of the metaphor was motivated with an emerging need of communicating important technical concepts (e.g., refactoring) to the non-technical stakeholders [3].

An important feature of the TD phenomenon, rooted in its nature, is the inherent temporal delay of TD effects manifestations. Therefore, the most valuable lessons to learn are from practitioners who have been actively involved with managing TD.

In this article, we focus on TD-experienced practitioners, and how they conceptualize key TD constructs. We used a family of surveys (see side box A) to collect feedback from industry experts. The most notable features of this report include:

- Analysis of 168 responses from practitioners who had experiences with managing TD, of those 74% having a technical role and 26% a non-technical role.
- Presentation of a model for conceptualizing TD causes, effects, payment practices and payment avoidance reasons with prioritization schemas for technical and non-technical roles. In total, 25 high-priority factors are extracted and explained.
- Analysis of the alignment between two perspectives. Data indicates a high level of alignment between technical and non-technical perspectives of TD causes and payment practices while the perspectives of TD effects and avoidance reasons are significantly less aligned.

Findings presented in this report are different from and complement the majority of TD management approaches that are focused on identification, prioritization, monitoring and mitigation of TD infected software artifacts [1]. We provide a conceptual model that takes into consideration the contextual factors and presents them in a form of prioritized lists of the most significant factors. Practitioners can utilize the findings and information presented graphically (Figure 1) for the purpose of coping with contextual factors surrounding the TD management process.

UBIQUITOUS AWARENESS ABOUT TECHNICAL DEBT

Ward Cunningham introduced the TD metaphor in the 1990s by making an analogy between *internal software quality* aspects and financial debt. While the metaphor is easy to comprehend, the underlying phenomenon remains elusive, multifaceted, and pervasive. For many years, different aspects of internal software quality were studied under different names [4]: software maintenance, evolution, aging, decay, re-engineering, sustainability, and so on. Later, it became clear that all those aspects are addressing the same phenomenon that is aptly described as technical debt [5].

Years of TD research and practice marked significant changes in ways how the industry deals with the TD phenomenon. Now, we understand that a more holistic approach is needed, that various code analysis tools are useful, and that context is a major factor when assessing and prioritizing TD [1], [6].

However, undeniably the single most important achievement is the increased awareness about TD within industry practitioners. In our research, we observed that seven out of ten practitioners are aware of TD and its consequences (see side box A). Even though some of them are not using sophisticated tools or techniques for TD management, the existing awareness about TD is encouraging.

CONCEPTUALIZING THE ELUSIVENESS OF THE TD PHENOMENON AND THE INSIGHTD SURVEY

The nature of the TD phenomenon is elusive and often tacit, which represents a challenge not only for research, but also for the industry. Therefore, we used a survey instrument (see side box A) that is designed using a few constructs understandable to various roles involved with software development. Those constructs are TD cause, TD effect, TD payment practice, and TD payment avoidance reason.

Preconditions that lead to the injection of TD are modeled as *causes*. For example, some often cited causes are lack of time or lack of resources [1], [2]. Note that causes stand for the stakeholders' comprehension of triggers that can lead to biased decisions toward TD.

Manifestations of TD are commonly known as *effects* or symptoms. Cognition about effects helps stakeholders to reason about potential problems that can occur as consequences of injected debt. Such reasoning about TD determines further actions regarding debt removal.

Although it is hard to quantify the impact of negative effects, some studies provide estimates of that impact, e.g., between 2USD and 20USD per line of source code of TD [7], or 25% of development effort is spent on TD caused issues [8].

Once TD effects become visible, a decision needs to be made whether *to pay*, i.e., to remediate, the effects, or to accept them and to avoid any corrective actions due to objective *reasons*.

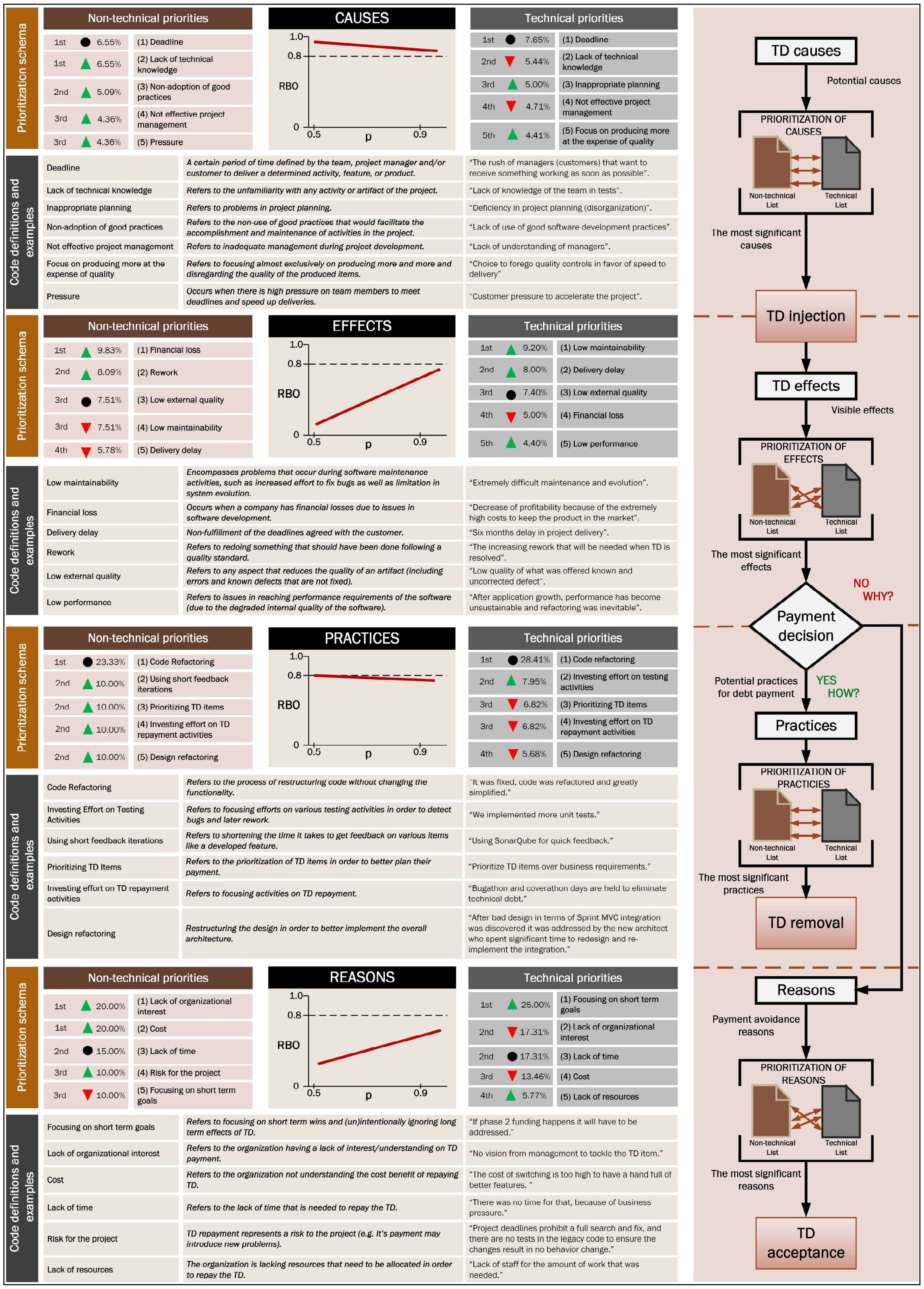
PRIORITIZATION SCHEMA FOR TD CONSTRUCTS

There is a multitude of factors that lead to favorable situations for TD injection, as well as many factors that impact a decision to remediate the injected debt or to accept it. Some factors are unique and very specific, while others can occur in different project contexts. By prioritizing TD constructs, we extract more common factors that represent generalizable knowledge about the TD phenomenon.

Furthermore, comparing technical and non-technical prioritization schemas gives us insight into the level of alignment of those two groups' perspectives regarding identified factors. Consequently, it is expected that better alignment leads to better communication with less friction between groups.

Figure 1 depicts the process of injecting and handling the injected TD. Prior to the injection of TD, potential causes can build-up a situation that results with newly created TD. Ideally, causes can be recognized before debt injection. This would require active monitoring of causes, by all involved practitioners. From Figure 1, practitioners can be informed about the most significant causes, either from technical or non-technical perspectives. The priorities indicate which causes are more serious with respect to TD injection.

Once TD is present in the project, a number of symptoms can be identified. Note that symptoms or effects may be considered as an indication of a present TD. Special attention is advised for effects with higher priorities. According to TD-experienced practitioners, higher priority effects indicate a higher probability of having TD in the project. The effects prioritization schema can be


CAUSES
1.0
0.8
RBO
0
0.5
p
0.9
EFFECTS
PRACTICES
REASONS
TD causes
Potential causes
PRIORITIZATION OF CAUSES
Non-technical List
Technical List
The most significant causes
TD injection
TD effects
Visible effects
PRIORITIZATION OF EFFECTS
The most significant effects
NO WHY?
Payment decision
Potential practices for debt payment
YES HOW?
Practices
PRIORITIZATION OF PRACTICES
The most significant practices
TD removal
Reasons
Payment avoidance reasons
PRIORITIZATION OF REASONS
The most significant reasons
TD acceptance

Prioritization schema — CAUSES

Non-technical priorities				Technical priorities			
1st	●	6.55%	(1) Deadline	1st	●	7.65%	(1) Deadline
1st	▲	6.55%	(2) Lack of technical knowledge	2nd	▼	5.44%	(2) Lack of technical knowledge
2nd	▲	5.09%	(3) Non-adoption of good practices	3rd	▲	5.00%	(3) Inappropriate planning
3rd	▲	4.36%	(4) Not effective project management	4th	▼	4.71%	(4) Not effective project management
3rd	▲	4.36%	(5) Pressure	5th	▲	4.41%	(5) Focus on producing more at the expense of quality

Code definitions and examples

Deadline	*A certain period of time defined by the team, project manager and/or customer to deliver a determined activity, feature, or product.*	"The rush of managers (customers) that want to receive something working as soon as possible".
Lack of technical knowledge	*Refers to the unfamiliarity with any activity or artifact of the project.*	"Lack of knowledge of the team in tests".
Inappropriate planning	*Refers to problems in project planning.*	"Deficiency in project planning (disorganization)".
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.*	"Lack of use of good software development practices".
Not effective project management	*Refers to inadequate management during project development.*	"Lack of understanding of managers".
Focus on producing more at the expense of quality	*Refers to focusing almost exclusively on producing more and more and disregarding the quality of the produced items.*	"Choice to forego quality controls in favor of speed to delivery"
Pressure	*Occurs when there is high pressure on team members to meet deadlines and speed up deliveries.*	"Customer pressure to accelerate the project".

Prioritization schema — EFFECTS

Non-technical priorities				Technical priorities			
1st	▲	9.83%	(1) Financial loss	1st	▲	9.20%	(1) Low maintainability
2nd	▲	8.09%	(2) Rework	2nd	▲	8.00%	(2) Delivery delay
3rd	●	7.51%	(3) Low external quality	3rd	●	7.40%	(3) Low external quality
3rd	▼	7.51%	(4) Low maintainability	4th	▼	5.00%	(4) Financial loss
4th	▼	5.78%	(5) Delivery delay	5th	▲	4.40%	(5) Low performance

Code definitions and examples

Low maintainability	*Encompasses problems that occur during software maintenance activities, such as increased effort to fix bugs as well as limitation in system evolution.*	"Extremely difficult maintenance and evolution".
Financial loss	*Occurs when a company has financial losses due to issues in software development.*	"Decrease of profitability because of the extremely high costs to keep the product in the market".
Delivery delay	*Non-fulfillment of the deadlines agreed with the customer.*	"Six months delay in project delivery".
Rework	*Refers to redoing something that should have been done following a quality standard.*	"The increasing rework that will be needed when TD is resolved".
Low external quality	*Refers to any aspect that reduces the quality of an artifact (including errors and known defects that are not fixed).*	"Low quality of what was offered known and uncorrected defect".
Low performance	*Refers to issues in reaching performance requirements of the software (due to the degraded internal quality of the software).*	"After application growth, performance has become unsustainable and refactoring was inevitable".

Prioritization schema — PRACTICES

Non-technical priorities				Technical priorities			
1st	●	23.33%	(1) Code Refactoring	1st	●	28.41%	(1) Code refactoring
2nd	▲	10.00%	(2) Using short feedback iterations	2nd	▲	7.95%	(2) Investing effort on testing activities
2nd	▲	10.00%	(3) Prioritizing TD items	3rd	▼	6.82%	(3) Prioritizing TD items
2nd	▲	10.00%	(4) Investing effort on TD repayment activities	3rd	▼	6.82%	(4) Investing effort on TD repayment activities
2nd	▲	10.00%	(5) Design refactoring	4th	▼	5.68%	(5) Design refactoring

Code definitions and examples

Code Refactoring	*Refers to the process of restructuring code without changing the functionality.*	"It was fixed, code was refactored and greatly simplified."
Investing Effort on Testing Activities	*Refers to focusing efforts on various testing activities in order to detect bugs and later rework.*	"We implemented more unit tests."
Using short feedback iterations	*Refers to shortening the time it takes to get feedback on various items like a developed feature.*	"Using SonarQube for quick feedback."
Prioritizing TD Items	*Refers to the prioritization of TD items in order to better plan their payment.*	"Prioritize TD items over business requirements."
Investing effort on TD repayment activities	*Refers to focusing activities on TD repayment.*	"Bugathon and coverathon days are held to eliminate technical debt."
Design refactoring	*Restructuring the design in order to better implement the overall architecture.*	"After bad design in terms of Sprint MVC integration was discovered it was addressed by the new architect who spent significant time to redesign and re-implement the integration."

Prioritization schema — REASONS

Non-technical priorities				Technical priorities			
1st	▲	20.00%	(1) Lack of organizational interest	1st	▲	25.00%	(1) Focusing on short term goals
1st	▲	20.00%	(2) Cost	2nd	▼	17.31%	(2) Lack of organizational interest
2nd	●	15.00%	(3) Lack of time	2nd	●	17.31%	(3) Lack of time
3rd	▲	10.00%	(4) Risk for the project	3rd	▼	13.46%	(4) Cost
3rd	▼	10.00%	(5) Focusing on short term goals	4th	▲	5.77%	(5) Lack of resources

Code definitions and examples

Focusing on short term goals	*Refers to focusing on short term wins and (un)intentionally ignoring long term effects of TD.*	"If phase 2 funding happens it will have to be addressed."
Lack of organizational interest	*Refers to the organization having a lack of interest/understanding on TD payment.*	"No vision from management to tackle the TD item."
Cost	*Refers to the organization not understanding the cost benefit of repaying TD.*	"The cost of switching is too high to have a hand full of better features. "
Lack of time	*Refers to the lack of time that is needed to repay the TD.*	"There was no time for that, because of business pressure."
Risk for the project	*TD repayment represents a risk to the project (e.g. It's payment may introduce new problems).*	"Project deadlines prohibit a full search and fix, and there are no tests in the legacy code to ensure the changes result in no behavior change."
Lack of resources	*The organization is lacking resources that need to be allocated in order to repay the TD.*	"Lack of staff for the amount of work that was needed."

Figure 1. Conceptual model of TD injection and acting upon it during software development. Technical and non-technical perspectives are modeled with the top-5 ranked lists of TD causes, effects, payment practices, and payment avoidance reasons. In addition, RBO stats is calculated for each TD construct.

used by practitioners to assess the project situation before any specific TD issues are identified.

At some point a decision is made to remediate TD issues. The decision is particularly interesting because the decision process reveals the level of organizational maturity towards managing TD. For example, in some organizations a form of cost-benefit analysis will be used to argument the decision, while in others no decision will be made explicitly, i.e., implicitly TD is accepted without clear reasoning behind such a decision.

Practitioners who are dealing with TD, can be in an unenviable position between business managers who do not perceive TD as an issue and colleagues who expect some action to be taken on the identified TD. Prioritization schemas for TD practices and payment avoidance reasons (Figure 1) can help by suggesting the most effective TD payment practices and providing the most significant avoidance reasons.

Especially, the list of payment avoidance reasons, can be used as a powerful tool to objectively prepare and argument TD payment decisions. For example, if a consensus is made to deal with TD, then the practitioners should address all major reasons in advance by taking actions to neutralize them. This can include dedicating funding, time, resources or altering short term goals. By doing so, the success of dealing with TD will be increased.

Alignment and Misalignment of Technical and Non-Technical Perspectives

Alignment of technical and non-technical prioritization schemas was assessed using the RBO method [9]. RBO results revealed that the causes and payment practices schemas are aligned ($RBO \gtrapprox 0.8$) while the effects and payment avoidance reasons schemas differ ($RBO \lessapprox 0.8$) and are thus seen as misaligned. To calculate the similarity, RBO uses a list of identified factors ranked by frequencies. Top five of these factors are presented for each TD construct in Figure 1, while the diagram between these lists shows the comparison result of all identified factors. The resulting curve in the diagrams shows the degree of similarity, or alignment between the lists depending on the number of top items that are emphasized ($0 < p < 1$). Note that the factor's rank is marked with numbers, while the factor's priority is presented in dedicated blocks beside the priority symbol and the ratio of the factor in all identified factors.

For example, if we look at the schema for TD practices (Figure 1), *code refactoring* has priority of 1^{st} at technical and non-technical lists and is thus marked with • at both lists. Furthermore, the remaining four items at the non-technical list have the same priority (2^{nd}), because they have the same percentage (10.00%) of all citations in the corresponding group. Also, indicators are useful to compare priorities, e.g., *design refactoring* at non-technical list has higher priority than it has at technical list and is marked with △ at non-technical list and with ▽ at technical list.

Alignment of perspectives on TD causes and payment practices

Technical and non-technical practitioners conceptualize TD causes and payment practices in the same way despite their different perspectives. We observe a high consensus about *pressure* (any form of pressure, e.g., time, productivity) as the cause with the greatest priority, while the *lack of technical knowledge* cause is trailing close and seems to have a greater priority from the non-technical perspective. Interestingly, this cause, although the name suggests it, is not specific only to the technical group. The non-technical people also use this cause to express, e.g., unfamiliarity with requirements management tools or business analysis techniques.

Regarding payment practices, *code refactoring* has the largest priority in both perspectives. It is encouraging to see that non-technical people understand the significance of code refactoring. According to our study, we conjecture that this practice is no longer difficult to explain and communicate with non-technical people. Ten years ago, refactoring was an exemplar technical practice that was difficult to explain to non-technical people [3].

Misalignment of technical and non-technical perspectives on TD effects and payment avoidance reasons

Perception of effects and non-payment reasons are far less aligned among the two groups. From this misalignment, however, interesting commonalities can be observed. First, there were no significant unique instances of TD effects or non-payment reasons that are specific for the one of the perspectives. This suggests that different roles have a very specific perception of the importance of the effects and payment avoidance reasons.

Second, the reasons for avoiding TD payment, with the largest priority, in both groups can be tied to *organizational issues* and therefore fall into the management domain. Technical people seem to understand that the causes and reasons for non-payments are also non-technical in nature.

The identified prioritization schema often contains identical factors, but these factors do not necessarily have the identical meaning in technical and non-technical contexts. For example, in the non-technical context, *Lack of technical knowledge* can denote the need for practitioners skilled in requirements management tools, whereas in the technical context, the need for skilled unit testers. On the other hand, the meaning of *code refactoring* is the same, regardless of the context. In the first case, the factor is interpreted with respect to the artifacts that are specific to groups, thus requires different treatment. In the second, the factor interpretation remained the same in both groups as the factor is tied to the same artifact. Highlighting on interpretation differences can contribute toward better communication and reasoning about TD constructs.

In the years to come, we need to see more work done on approaches for *de-pressuring* the software development process to alleviate negative consequences of the pressure on TD injection. The importance of such approaches needs to be understood on an organizational and managerial level.

The InsighTD project team is committed to investigating the TD phenomenon, an interested reader can be further informed at project's website (footnote 1). At the website besides the research roadmap and publications, guidelines for joining the initiative can be found.

Appendix

Sidebox 1. Methodological approach

This study is part of InsighTD[1]—a global initiative to investigate TD from different perspectives. A family of surveys was used to collect the data, while the survey design[2] allowed participants to report multiple instances of TD constructs, TD management related data, as well as the demographic data. Participant selection followed the convenience sampling approach [10] resulting with 653 responses. This study relied on answers from 168 participants out of which 124 technical and 44 non-technical. These participants stated to be experienced in software development and experienced with TD management (Figure 2).

Collected data was analyzed using both qualitative and quantitative research methods. Qualitative coding was used to analyze open-type questions i.e. to identify TD constructs [11]. Descriptive statistics was primarily used to characterize the demographics (closed-type questions) and the identified TD constructs. Rank-biased overlap (RBO) method[3] was used to compare the technical and non-technical lists.

To convey the *internal validity*, each research team followed the common analysis procedure that involved iteration, review, and decision by consensus. Additionally, each team evolved the common code-schema derived based on previous replications. The *external validity* is addressed by collecting data from industry practitioners from different countries, work environments and roles. We acknowledge that the unbalanced number of technical and non-technical responses used in this study may challenge the generalizability of the findings. Still, since the RBO method is resilient to frequency changes (footnote 3) and that the unbalance is inherent in the target population, the sample is seen as suitable for this study.

REFERENCES

1. N. Rios, M. G. de Mendonça Neto, and R. O. Spínola, "A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners," *Information and Software Technology*, vol. 102, 2018.

[1] http://www.td-survey.com
[2] The survey instrument: https://bit.ly/2SCOM9S
[3] RBO analysis report is accessible at https://bit.ly/3qjEZSK

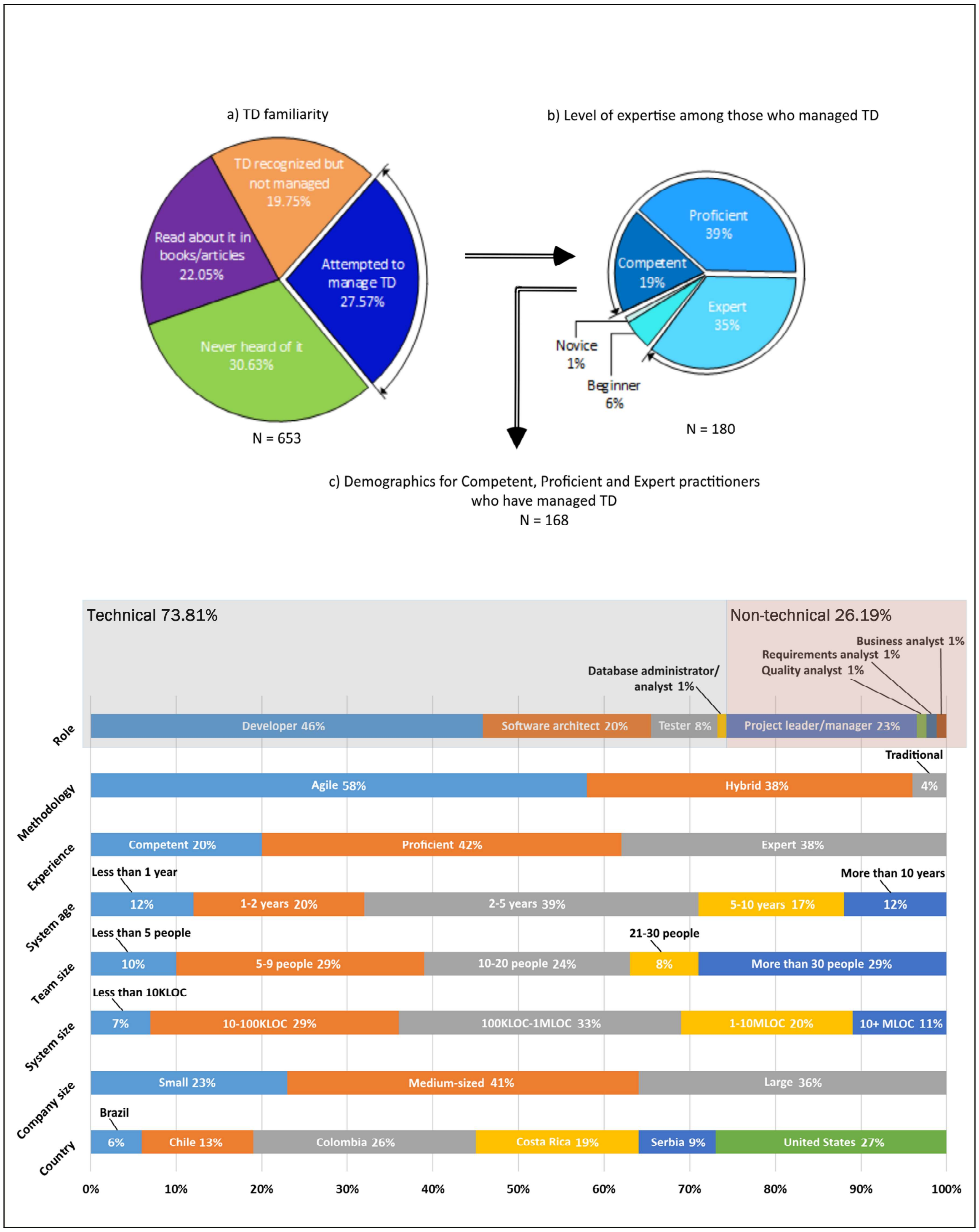


Figure 2. Respondents TD familiarity. Level of expertise of respondents that managed TD. Technical and non-technical structure of experienced respondents that managed TD.

2. R. Ramač, V. Mandić, N. Taušan, N. Rios, M. G. de Mendonca Neto, C. Seaman, and R. O. Spínola, "Common causes and effects of technical debt in Serbian IT: InsighTD survey replication," in *2020 46th Euromicro Conference on Software Engineering and Advanced Applications*, 2020.
3. P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, vol. 29, no. 6, 2012.
4. P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," *IEEE Software*, vol. 33, no. 1, 2016.
5. P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162)," *Dagstuhl Reports*, vol. 6, no. 4, 2016.
6. V. Lenarduzzi, T. Besker, D. Taibi, A. Martini, and F. Arcelli Fontana, "A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools," *Journal of Systems and Software*, vol. 171, 2021.
7. B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the principal of an application's technical debt," *IEEE Software*, vol. 29, no. 6, 2012.
8. T. Besker, A. Martini, and J. Bosch, "Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work," *Journal of Systems and Software*, vol. 156, 2019.
9. W. Webber, A. Moffat, and J. Zobel, "A similarity measure for indefinite rankings," *ACM Trans. Inf. Syst.*, vol. 28, no. 4, Nov. 2010.
10. C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslén, *Experimentation in software engineering*. Springer Science & Business Media, 2012.
11. N. Rios, R. O. Spínola, M. Mendonça, and C. Seaman, "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*, vol. 25, no. 5, 2020.