Technical Debt Tools: A Systematic Mapping Study

Diego Saraiva[a], José Gameleira Neto[b], Uirá Kulesza[c], Guilherme Freitas[d], Rodrigo Reboucas[e] and Roberta Coelho[f]

Department of Informatics and Applied Mathematics, Federal University of Rio Grande do Norte, Natal, Brazil

Keywords: Systematic Mapping Study, Technical Debt, Technical Debt Management, Tools.

Abstract: *Context*: The concept of technical debt is a metaphor that contextualizes problems faced during software evolution that reflect technical compromises in tasks that are not carried out adequately during their development - they can yield short-term benefit to the project in terms of increased productivity and lower cost, but that may have to be paid off with interest later. *Objective*: This work investigates the current state of the art of technical debt tools by identifying which activities, functionalities and kind of technical debt are handled by existing tools that support the technical debt management in software projects. *Method*: A systematic mapping study is performed to identify and analyze available tools for managing technical debt based on a set of five research questions. *Results*: The work contributes with (i) a systematic mappping of current research on the field, (ii) a highlight of the most referenced tools, their main characteristics, their supported technical debt types and activities, and (iii) a discussion of emerging findings and implications for future research. *Conclusions*: Our study identified 50 TD tools where 42 of them are new tools, and 8 tools extend an existing one. Most of the tools address technical debt related to code, design, and/or architecture artifacts. Besides, the different TD management activities received different levels of attention. For example, TD identification is supported by 80% of the tools, whereas 30% of them handle the TD documentation activity. Tools that deal with TD identification and measurement activities are still predominant. However, we observed that recent tools focusing on TD prevention, replacement, and prioritization activities represent emergent research trends.

1 INTRODUCTION

The concept of technical debt (TD) is a metaphor used to represent technical compromises that can yield short-term benefit to the project in terms of increased productivity and lower cost, which may affect the software maintenance and evolution (Avgeriou et al., 2016).

In real-life software development, the presence of technical debt is inevitable (Martini et al., 2015) and even desirable to achieve short-term benefits. It can be profitable (Allman, 2012) if the cost of the technical debt is manageable. Therefore, it is imperative to keep the accumulated debts under control. In this context, the purpose of technical debt management (TDM) is to support informed decision-making about the need to mitigate a TD item and the most suitable time to do this (Guo et al., 2016). It has been emerged as a new research area over the last years. It consists of a series of activities that prevent the creation of an unwanted technical debt or allows dealing with existing debt to keep it below the permissible limit. However, the efficient management of TD requires the adoption of tools.

Recently, several tools to manage technical debt in software projects have been proposed by academia and industry. In this context, we identified that an in-depth characterization of TD tools is missing in the literature. Existing systematic reviews and mapping studies focus on specific aspects of technical debt. (Li et al., 2015) presents a systematic mapping study that aims to understand the state of research on TD management. Their study found that TD was classified into 10 types, 8 TDM activities were identified, and 29 tools for TDM were collected.

(Lenarduzzi et al., 2019) conducted a systematic literature review on technical debt prioritization. They identified 12 tools related to TD prioritization.

[a] https://orcid.org/0000-0002-2512-6692
[b] https://orcid.org/0000-0002-7184-1187
[c] https://orcid.org/0000-0002-5467-6458
[d] https://orcid.org/0000-0003-0272-0071
[e] https://orcid.org/0000-0001-7829-9768
[f] https://orcid.org/0000-0003-0001-435X

The main purpose of their review is to understand how the TD is prioritized in software organizations and which approaches have been proposed. The authors highlight a lack of a solid, validated, and widely used set of tools specifically for TD prioritization. They did not detail the tools' features, limiting to present a brief description of them, the papers that cite the tools, and the context of their usage.

(Avgeriou et al., 2020) provide an overview of the current landscape of existing TD measurement tools. They analyze the features and popularity of these tools, the existing empirical evidence on their accuracy, and highlight current shortcomings. The authors consider initially 26 tools and filtered them to select only 9 for a complete analysis. Differently from this work, they focus strictly on analysing technical debt measurement tools.

In order to mitigate the literature gap, we present an overview of the currently available tools for technical debt. The purpose of this paper is to characterize the state of the art of TD tools. To achieve this goal, we have conducted a systematic mapping study of the literature in the area. We synthesize the data obtained to produce a clear overview of existing TD tools. Our mapping represent a frame of reference for both: (i) the academia to identify research opportunities; and (ii) the industry to understand and to know existing research results to possibly transfer and mature TD tools to practice.

The remainder of this paper is structured as follows: Section 2 details he methodology of the mapping study. Section 3 presents the study results and their implications to researchers and practitioners. Section 4 presents the threats to validity of the study. Section 5 discusses related work. Finally, Section 6 presents the paper conclusions.

2 STUDY DESIGN

We conducted a systematic mapping study based on well-established guidelines in software engineering (Petersen et al., 2015) (Kitchenham and Charters, 2007). The purpose of this study is to provide an overview of the currently available tools for managing technical debt for professionals and researchers. In particular, we focus on the following research questions (RQs):

RQ1: What Are the Publication Trends of Research Studies about TD Tools?

RQ2: What Are the Main Characteristics of the Tools?

RQ3: What Are the TD Tools Reported in the Literature?

RQ4: Which TD Types and Activities Are Addressed by the Proposed Tools?

RQ5: What Kind of Studies Have Been Conducted to Evaluate TD Tools?

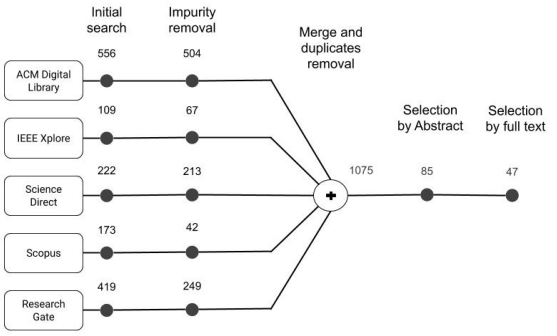

Figure 1: Overview of the search and selection process.

Figure 1 shows the search and selection process of the systematic mapping study. The search process of our study considered journal, conference, book chapter and workshop papers indexed in the following digital libraries: ACM, IEEE, Science Direct, Research Gate and Scopus. These databases were selected because they are the largest and most complete scientific databases in software engineering (Petersen et al., 2015) (Kitchenham and Charters, 2007). They have been recognised as being an effective means to conduct systematic literature studies in software engineering (Petersen et al., 2015). Our search string is shown in the Listing 1. For consistency, the search string has been applied to the title, abstract, and keywords of papers in the selected electronic databases and indexing systems. The search in the selected databases was conducted in February 05 - 2020.

Listing 1: Search string used for automatic research studies.

```
("Technical Debt") AND
("Tool" OR "Software Solution")
```

Based on our research questions, we defined inclusion and exclusion criteria. These criteria are essential to identify relevant primary studies that answer the RQs. We devised the following inclusion criteria: (i) the paper was written in English, (ii) the paper was published in conference proceedings, workshop proceedings, book chapters or journal; and (iii) the paper proposes, extends or evaluates one or more TD tools. We retrieved 1479 primary papers in the initial search. Next, in the merge and removal step, we removed invalid results returned by the execution of the search query, given that some results are duplicates or

are not research papers (patents, standards, etc). We include papers that address the inclusion criteria (i) and (ii) for each database. Finally, we join all the papers in a single set having as a result of total of 1075 papers. However, only 85 candidate papers were obtained after applying the inclusion criteria based upon title and abstract in the stage of selection by abstract and applying the criterion (iii). Finally, we selected only 47 primary papers after reading the full paper. Additional information about the study protocol and its results can be found in (td-tools study, 2020).

3 RESULTS AND DISCUSSION

In this section, we discuss the answers to our RQs presented in Section 2. In each case, we highlight the utility of these results for researchers and practitioners. Additional information about the study results can be found in (td-tools study, 2020).

3.1 RQ1: What Are the Publication Trends of Research Studies about TD Tools?

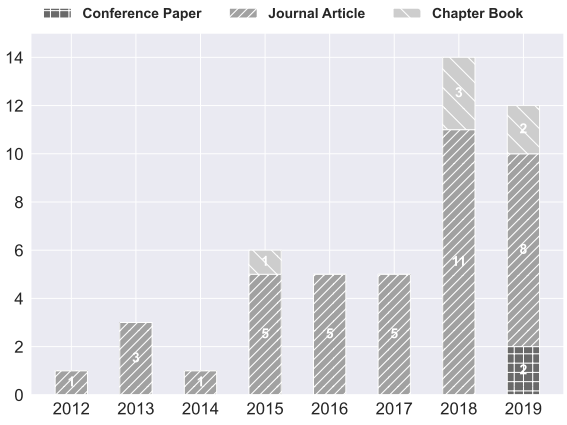

Figure 2: Distribution of selected studies over publication types.

The purpose of this research question is to provide an overview of the number and types of publications on the topic during the last years. Figure 2 presents the distribution of publications on technical debt tools over the years. We can see that conference papers are the main kind of publication selected in our study with 83% (39/47) of the papers. The journal papers represent 12.8% (6/47) of the selected studies. Finally, only 4.3% (2/47) fall into the category of the book chapter. The high number of conference and journal papers indicate that technical debt tools is a trending research topic and indicates researchers target more scientifically-rewarding publication when working with tools to support technical debt. Figure 2 emphasizes an explicit confirmation of the scientific interest on technical debt tools in the years 2015 through 2019. Since 2015, there are at least five studies per year, which represents a good increment compared with the years before 2015. One reason for that could be that technical debt becomes a more known and popular concept, increasing the interest of incorporating TD tools in the daily practice of software development. The most recurrent places for publications about TD tools were: (i) the *International Conference on Technical Debt* (TechDebt) and (ii) the *International Workshop on Managing Technical Debt* (MTD) with a total of 15 papers. The TechDebt conference is an evolution of the MTD workshop. Secondly, the *Euromicro Conference on Software Engineering and Advanced Applications* (SEAA) with 5 papers. Following by textitSymposium on Applied Computing (ACM) and *Journal of the Brazilian Computer Society* with 2 papers. We can observe a fragmentation in terms of publication venues, where research on TDM tools is spread across 20 venues.

The results of this research question help researchers and practitioners: (i) to estimate the enthusiasm of scientific engagement on technical debt tools; (ii) to identify the academic venues where related papers about technical debt tools are published; and (iii) to identify the academic venues where new results about technical debt tools may be better received and recognized by the scientific community.

3.2 RQ2: What Are the Main Characteristics of the TD Tools?

In this subsection, we present the obtained results when analyzing the main characteristics of the selected tools. We analyzed the following aspects of the selected tools: (I) main purpose; (II) target languages; and (III) the tool was developed as standalone or it represents an extension of an existing tool. Table 1 presents the identified tools considering their main purpose.

Concerning the primary purpose of the tools, we classify them using a categorization, inspired and adapted from the Li et al. study (Li et al., 2015). Figure 3 shows the categorization, which aggregates the background approaches that support the investigated TD tools. It also presents the distribution of tools considering the different categories for their main purpose.

Our study identified that 28% (14/50) of the TD tools aims of quantifying code metrics. These

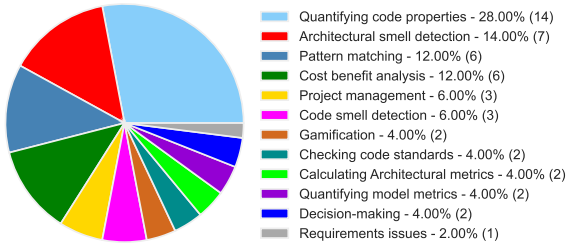

Figure 3: Distribution of tools over the categories.

tools quantify TD using code metrics, and/or analyze source code to identify coding rules violations, detect potential bugs, and many other specific code issues. Architectural smell detection represents the second most common category, with a total of 14% (7/50) of the tools. This result was expected because architectural problems, such as modularity violations, are a common source of the perception of technical debt in software industry (Apa et al., 2020). These results indicate that most of the existing TD tools focus on the quality assurance over source code and related services, i.e. quantifying code metrics, architectural smell, code smell, and checking code standards. This mainly occurs because there is a consolidated set of metrics to the measurement of the source code quality; therefore, static code analysis tools can be used to extract TD indicators. For example, by using source code analysis tools, it is possible to identify refactoring opportunities, detect security vulnerabilities, highlight performance bottlenecks, and identify bad programming practices, such as code smells. Another potential explanation for emphasis of the tools in the source code is that the body of knowledge in technical debt is still consolidating on software development (Alves et al., 2016). Issues related to source code are more visible to developers as they are more focused on implementation artifacts.

It is also possible to observe that there are technical debts throughout many software artifacts of different stages of software development. However, few tools utilise non-code artifacts as input. More specifically, only 4% (2 tools) of them apply a model-driven approach to calculate technical debt related to model elements, and only 1 tool addresses TD related to requirements specifications. These tools have the advantage of acting on early phases of software development, avoiding technical debt later consequences in the projects (Giraldo and Osorio, 2017).

The literature reports that developers often lack the motivation to address technical debt (Foucault et al., 2018). In this context, some tools apply gamification techniques to manage TD providing suggestions for developers on where to focus their effort, visualizations to track technical debt activities, and

Table 1: Technical Debt Tools by Main Purpose.

Main Purpose	Tool Name
Architectural smell detection	Arcan, Arcan C++, Designite, Designite Java, DV8, Lattix, Sonargraph
Quantifying Architectural metrics	Dependency Tool, Structure 101
Quantifying code metrics	CBRI Calculation, Code Analysis, CodeScene, DBCritics, DeepSource, inFusion, Jacoco, ProDebt, Sonarcloud, SonarQube, Squore, Teamscale, TEDMA, VisminerTD
Quantifying model metrics	EMF-SonarQube, BPMN-spector
Code smell detection	CodeVizard, JSpIRIT, Ndepend, FindBugs, checkstyle
Cost benefit analysis	AnaConDebt, CAST, FITTED, JCaliper, MIND, TD Tool
Decision-making	Georgios Tool, TD-Tracker Tool
Gamification	Build Game, Themis
Pattern matching	SAApy, DebtFlag, Debtgrep, eXcomment, MAT, SATD Detector
Project management	Hansoft, Jira, Redmine
Requirements issues	Requirements Specification Tool

stimulating TD prevention. However, this kind of initiative is in an early phase since only 4% (2/50) of the tools implement this approach.

The pattern matching approaches analyze the source code to identify patterns that characterize technical debt. 12% (6/50) of the selected tools implement a solution based on pattern matching. Most of pattern matching tools (3/6) are intended to recognize self-admitted technical debt (SATD) from source code comments. This kind of technical debt considers that they are intentionally introduced and admitted by developers.

The above-cited approaches, such as code smell detection, have been developed to identify particular TD types to detect components that need to repay debt. However, an important issue is to provide more guidance for decisions about whether or not to pay off particular TD instances at a given point in time (Guo et al., 2016) (Seaman and Guo, 2011). The cost-benefit analysis tools provide approximation methods for helping in the prioritization of resources and efforts concerning refactoring strategies. The tools in this category represent 12% of the total. For example, the AnaconDebt tool allows you to assess the prin-

cipal (cost of refactoring) and interest (current and future extra costs) of the technical debt. Unfortunately, the AnaconDebt is not open source and the formula used to estimate the principal and interest are confidential. Another highlight is JCaliper. This tool applies a local search algorithm to obtain a near-optimum design for the software. It also proposes TD repayment actions (a sequence of refactorings) to reach it. The distance quantifies the difference in the selected fitness function and reflects the architectural quality of the examined system. The distance also translates to a number of refactorings required to convert the actual system to the corresponding optimum one.

A correlated category is decision-making tools, 4% (2/50) of the tools, that concentrates tools with a reflection on the value of the TD from a more business-driven approach taking into account other aspects besides project-related benefit (Riegel and Doerr, 2015). In this group, we call attention to the TD Tracker tool. It presents an approach to create an integrated catalogue as metadata from different software development tasks in order to register technical debt properties and support managers in the decision process.

Finally, the project management category aggregates 6% (3/51) of the tools. The tools in this group do not address any specific TD type, focusing on tracking and monitoring technical debt. In this category, we found tools like Hansoft, Jira and Redmine.

Concerning the target languages of the tools, we have classified as language-specific the tools that are specific to one particular language (e.g., C++). A total of 56% (28/50) of the tools are language-specific, while the remaining 26% (13/50) of them are not language-specific, and 18% (9/50) tools do not address any specific language. The predominance of language-specific tools is not a good indicator because they can not be reused across different technologies and languages. Therefore, their applicability and portability in the future can be limited. However, language-specific tools have the advantage of being more tailored to the domain. They have the potential to address specific characteristics of the languages that allow better analysis. Our investigation concluded that Java with 58% (29/50) of them, C# with 22% (11/50), and C++ with 20% (10/50) are the most common target languages.

At last, we analyzed whether a solution is an extension of other existing one or is a novel solution. 84% (42/50) of the tools are new tools and are not based on other existing ones, while the remaining 16% (8/50) extends a previous solution. In this context, our research found a limited number of tools that provide extension mechanisms that allow third-party development. SonarQube and Arcan were the tools with most extensions available with three and two extensions, respectively. We considered as extensions since the simple addition of new features until derivation for new tools based existing tool.

3.3 RQ3: What Are the TD Tools Reported in the Literature?

The purpose of this research question is to analyze the state of the art of TD tools from two different aspects: (i) the explicit support to the technical debt concept; and (ii) the research type.

Previous work (Mamun et al., 2019) reported the lack of specialized tools to deal with the TD concept. The explicit usage of the TD concept in the tools helps to identify whether a particular tool aims to address technical debt management (TDM) issues. This work considers as a specialized tool those ones that model or implement explicitly the TD concept/abstraction. Otherwise, we classify them as a generic tool that is adapted to deal with TD issues. In our study, we found that 80% (40/50) of the tools provide explicit support to the technical debt concept, while only 20% (10/50) do not explicitly use the TD abstraction in the tool but are used to address some kind of TD reported in the respective paper.

To categorize the TD studies, we adapted the research types classification suggested by (Wieringa et al., 2005): (I) proposing new tools; (II) extending existing tools; or (III) evaluating existing tools. The proposition of new tools is present in 57% (27/47) of the selected papers, indicating that the TD tools are still in their maturing phase with new ones being proposed over the last years. There is a large number of researchers proposing their own solutions for either recurrent or specific problems. The evaluation of existing tools is the subject of 25% (12/47) papers, which represents the second most recurrent research strategy. This highlights the fact that researchers are looking for some level of evidence about their proposed tools by investigating and applying them in practice, and conducting evaluations to validate their claims. At the other end of the spectrum, the research about the extension of existing tools is performed in only 17% (8/47) of papers.

The results of this research question can help researchers and practitioners (i) to have an overview of the currently available tools for managing technical debt, and (ii) to quantify the degree of scientific interest on TD tools.

3.4 RQ4: Which TD Types and Activities Are Addressed by the Proposed Tools?

This research question investigates how the 50 TD tools identified in our systematic mapping study address existing kinds of technical debt and technical debt management (TDM) activities.

3.4.1 RQ 4.1: What Are Kinds of TD That the Tools Focus On?

This subsection discusses the TD types addressed by the selected tools. The present work points out that the main TD types addressed by tools deal with source code (60% - 30/50), architectural issues (40% - 20/50) and design issues (28% - 14/50). Previous studies (Rios et al., 2018) (Li et al., 2015) found similar results, meaning that researchers on technical debt tools have maintained their interest in these TD types over the last years. This trend is in line with the original definition of technical debt, which is heavily influenced by concepts coming from source code and related issues. As mentioned before, most of TD studies concern code mainly because there are several available tools providing useful information about code quality.

In compensation, there are many TD types (build, defect, requirements, infrastructure) that are addressed by a reduced number of tools (respectively, 2, 2, 2 and 1). One potential reason is that despite these TD types impact on the productivity of software development, they do not have a direct impact on the software quality as the code related TD types have. Our mapping study shows that there are gaps for future research in tools for supporting these TD types.

Our analysis also shows that some existing tools (3/50) do not focus on a specific TD type. For example, Hansoft[1] and Jira[2] are tools predominantly designed for software development project management. Hansoft allows recording technical debt in the form of a list or with a graph which is manually elaborated. In turn, JIRA provides support to register the technical debt items and assign them a priority score. Thus, these tools are mainly used for the technical debt management to explore the backlog. In comparison, TD-Tracker TD amount estimation besides technical debt management (Pavlič and Hliš, 2019). The reduced number of tools that address project management activities shows that there are opportunities for improving the existing ones to provide efficient management of technical debts.

[1]https://www.perforce.com/products/hansoft

[2]https://www.atlassian.com/software/jira

Our investigation shows that most of the tools, 60% (30/50) of them, are tailored to a particular TD type. On the other hand, 34% (17/50) of the tools are not dedicated to a particular TD type, such as SonarQube and Arcan. Finally, 6% (3/50) of the tools do not address any specific TD type, such as TD-Tracker. TD-Tracker implements an approach to tabulating and managing TD properties to support project managers in the decision process. It integrates with different TD identification tools to import technical debt already identified. This later kind of tool handles with TD concept in a generic way. It is worth to highlight that 16% (8/50) of tools are associated with two TD types, notably the TD tools related to code, architecture and design. In this group, we have, for example, Structure101 that address code and architectural TD, and Lattix that handles design and architectural TD. Finally, we found a huge gap on the TD tools concerning the support to versioning debt, once this is the unique one without a dedicated tool to support it. The versioning debt refers to problems in code versioning, such as to identify unnecessary code forks or the need to have multi-version support.

Comparing our results with previous work, we observed an increase of tools that focus on architectural TD over the last years. Li et al. (Li et al., 2015) mentioned that only 10% of the tools provide support to architectural TD against 40% found in our review. Although a great deal of theoretical work on the architectural aspects of TD has recently been produced, there was still a lack of TD tools to deal with architectural issues. However, this scenario has been changing in recent years. We can conjecture that this trend will continue over the next years due to its significant impact on the quality of software systems. Similarly, design TD has also received more attention over the last years. This reflects a better understanding of the impact of design debt on the quality of software systems.

3.4.2 RQ 4.2: Which TD Management Activities Do the Tools Focus On?

The results point out TD identification is widely supported by 80% (40/50) of the tools. TD measurement also received huge support from the tools with 64% (32/50) of them addressing this activity. We observe a strong interest of the existing tools to provide support to TD identification and measurement activities since 2015. However, comparing the growth rates between TD identification and TD measurement activities, we observe a small negative trend on TD identification, meaning that a growing number of research work are now focusing on TD measurement.

When we compared our results with the previous

ones found in literature, we recognise a spike in the communication activity support in the last year. Li et al. (Li et al., 2015) mentioned that only 28% of the tools provide support to communication, contrasting with 63% identified during our systematic mapping. Tools like AnaConDebt supports the creation of a TD-enhanced backlog, allowing the tracking of TD items, and thus facilitating their estimation and communication among the stakeholders. We can conjecture that this trend will continue over the next years since communication activities make the technical debts more visible and understandable to stakeholders allowing them to be discussed and managed appropriately. In line with this trend, we also identified an increase on the support to the TD documentation activity, indicating that researchers are studying and devising new approaches to address technical debts that are not directly related to source code. We identified that TD documentation is supported by 30% (15/50) of tools. For example, Teamscale applies code quality analyses to detect missing software documentation.

Regarding the TD prioritization activity, our mapping study found 13 tools, i.e. 23% of them, that handle this activity. However, most of these tools only provide means to assign priority to a given TD, thus only a few tools implement an approach to assign a priority to a TD automatically. Currently, there is no consensus on what are the critical factors and how to prioritize them. This context is mirrored in the fragmented support offered by the tools, leading to a lack of widely used and validated set of tools specific to TD prioritization (Lenarduzzi et al., 2019). Besides that, our work revealed a scarcity of approaches that account for cost, value, and resources constraint, and a lack of industry evaluation of this kind of tool. Nevertheless, this context reveals exciting gaps in the literature. We believe research on TD prioritization is still evolving and is a promising research direction.

Next, we have the prevention and repayment activities, both supported by 12% (6/50) of the tools. Although there are several strategies proposed to TD management in the literature, preventing the TD insertion with explicit actions is not yet common practice (Rios et al., 2018). This can be the reason for the relatively low number of tools that address this activity. However, interestingly, we can also observe that the scientific interest on prevention approaches has been increasing since 2015. We conjecture that this trend derives from the awareness of the technical debt cost and the consensus that TD prevention can occasionally be cheaper than its repayment. Moreover, prevention may also help other TD management activities, as well as help in catching inexperienced developer (Yli-Huumo et al., 2016). For example, Themis (Foucault et al., 2018) is a customized gamification tool that integrates with SonarQube and version control tools with the aim of identifying and measuring TD. Themis uses gamified features such as points, leaderboards, and challenges as a way to provide suggestions for developers on where to focus their effort, and visualizations for managers to track technical debt activities. The monitoring features provided by Themis become the technical debt more visible to the developers helping in the prevention of technical debts.

Regarding the repayment activity, it usually concerns resolving technical debt through techniques such as reengineering or refactoring. There are many challenges in applying refactorings to repay technical debt in large-scale industrial software projects (Suryanarayana et al., 2015). For example, it is hard to ensure that the behavior of the software is unchanged after the refactorings. It is not surprising that the number of tools addressing this concern is low. For example, JCaliper (Kouros et al., 2019) applies search-based software engineering techniques as a means of assessing TD and proposing a set of refactorings to address it. It performs local search algorithms to obtain a near-optimum solution and to propose TD repayment actions, automatically extracting the number, type and sequence of refactoring activities required to obtain the design without TD.

Our study also identified that 78% (39/50) of the tools are dedicated to more than one TD activity. The tools are usually associated with at least two different TD activities, like Arcan that deals with identification and measurement activities, or SATD that provides support to TD monitoring and identification. On the other hand, 22% (11/50) of the tools are specialized in just one TDM activity, like FITTED that only focuses on measurement.

Figure 4 presents the different level of attention received by each TDM activity and TD type. The number into the bubbles represents the number of tools associated with both a TD type and a TDM activity. Among the ten TD types, as mentioned before, the TD code is the most investigated in the selected papers. It is also the TD type that is approached for all TD activities, especially identification and measurement. Therefore, code is the most valuable source of information for technical debt tools. Most of the technical debt tools exploit static code analysis techniques, so this finding is in line with the expectations.

The RQ4 results are useful to inform practitioners what approaches they have addressing specific TDM activities, and also to help researchers to identify research gaps in approaches for various TDM activities.

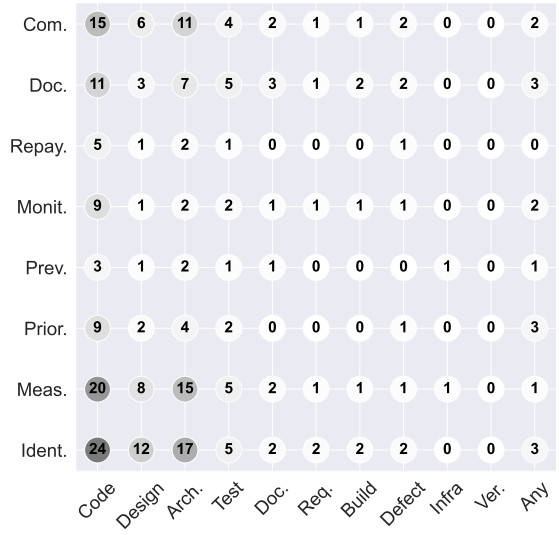

Figure 4: Relationship between TD types and TDM activities.

3.5 RQ5: What Kind of Studies Have Been Conducted to Evaluate TD Tools?

The motivation behind this research question is an evaluation of the potential for industrial adoption of existing TD tools. From a practitioner's point of view, it is important to have a reasonable level of confidence to use a given TD tool in software projects. In this context, there are different kinds of empirical studies that could be used to gain evidence about the feasibility and effectiveness of proposed tools. The application of the empirical paradigm to support an evaluation in software engineering is important because they contribute to a higher level of maturity of the tools and better acceptance in the software industry.

Wohlin et al. (Wohlin et al., 2012) classify the studies according to the research method used as follows: Case study, Survey (questionnaire, observation, interview), and Experiment. Besides, we have included a new category called Not reported; representing the papers that do not make explicit the research method of the evaluation.

Our work points out that most of them - 46.81% (22/47) - applied Case Study as the methodology to evaluate their research. The second most frequent study type was Survey, present in 21.28% (10/47) of the papers. The third most used study, the Experiment methodology was the least used to the evaluation of TD tools, present in only 14.89% (7/47) of the papers. Finally, 17.02% (8/47) of the studies do not present any formal or systematic evaluation. The literature reports that both academia and industry have significant interest in the TD tools (Li et al., 2015). It is important to emphasize that our study only assesses if the mentioned tools have some evaluation evidence of their usage based on the selected papers of our systematic review.

We also collected data on the type of research methodology of the studies by discriminating among qualitative, quantitative, or mixed analysis approaches. The data reports that 17% (8/47) of studies did not report any analysis approach. The most frequent type of analysis was qualitative, which is present in 44% (21/47) of selected papers. 36% (17/47) of them used a quantitative approach. Finally, we found that mixed analysis (qualitative/quantitative) was used only in 19% (9/47) of the studies.

We also identified that SonarQube (7 papers), CodeScene (3 papers), TeamScale (3 papers) and SonarGraph (3 papers) were the most evaluated tools by the literature.

4 THREATS TO VALIDITY

In this section, the threats to validity of our study are described:

Construct Validity. The procedure of our mapping study followed the guidelines for performing secondary studies proposed by Kitchenham et al. (Kitchenham and Charters, 2007). First of all, systematic mapping studies are known for not guaranteeing the inclusion of all the relevant works on the field. A possible lack of a set of keywords in the string search defined for the study can exclude some relevant papers. To deal with this threat, we used a broader search criterion to include a high number of related papers. Another problem can be the definition of what is a technical debt tool. We consider as TD tools the ones that the authors reported as dealing with TD concepts in their respective papers.

Internal Validity. In order to reduce this threat, the stages of selection of the studies and data extraction were carried out by three PhD students using a protocol rigorously defined. The results found by each one were tabulated and compared so that any kind of bias could be identified, and when in disagreement, the authors could debate and a consensus was reached.

External Validity. This threat concerns the generalizability of the obtained results (Wohlin et al., 2012). The most severe potential external threat to the validity of our study is our primary studies not being representative of state of the art on technical debt tools. To avoid it, we performed an automatic search in the five most popular electronic databases. We are reasonably confident about the construction of the search string

since the used terms are generic, allowing us to explore the field in a wide scope. This was reflected on the significant number of papers gathered during the process.

Conclusion Validity. Bias on the data extraction process may result in the inaccuracy of the extracted data items, which may affect the analysis and classification of the results of the selected studies. It was necessary to guarantee that the data extraction process was aligned with our research questions. We also mitigated potential threats related to conclusion validity by applying the following actions. First, the data items extracted in this mapping study were discussed among the researchers and agreement on the meaning of each data item was achieved. Second, a trial data extraction was performed among researchers, and disagreements on the results of the trial data extraction were discussed with two additional researchers to achieve a consensus. Finally, the data extraction results were checked by two researchers, and again disagreements were discussed and resolved with two additional researchers. All these actions improved the accuracy of the extracted data items.

5 RELATED WORK

Verdecchia et al. (Verdecchia et al., 2018) conduct a secondary study that focuses on the analysis of the literature related to architectural technical debt (ATD). The authors selected and inspected 47 primary studies to provide a characterization for ATD identification techniques in terms of publication trends, their characteristics, and their potential for industrial adoption. Our study differs from theirs by zooming out the analysis of TDM tools, not being restricted to only ATD related tools. Their work unveils some promising areas for future research on ATD, such as (i) the exploitation of the temporal dimension when identifying ATD; and (ii) the related resolution of ATD. The authors highlight that further industrial involvement when formulating, designing, and evaluating the ATD identification techniques is needed.

Lenarduzzi et al. (Lenarduzzi et al., 2019) conducted a systematic literature review about technical debt prioritization. Their work considered papers published before December 2018. The study was based on 37 selected studies, which represent the state of the art concerning approaches, factors, measures and tools used in practice or research to prioritize technical debt. They identified 7 tools that address the technical debt prioritization. The main outcome of their study is that there is no consensus on what are the important factors to prioritize TD and how to measure them. Their results report that code and architectural debt are by far the most investigated kind of debt when considering the prioritization. This trend was confirmed by our study, indicating that existing TD tools focus more on code and architectural issues. Another finding confirmed by their and our study is the lack of a solid, validated and widely used set of tools specific to TD prioritization.

Behutiye et al. (Behutiye et al., 2017) report the results of a systematic literature review that addressed the concept of technical debt in the agile software development context. The authors analyzed 38 papers out of 346 studies to pinpoint research areas of interest, TD cause and effect classifications, management strategies and tools. They mentioned the DebtFlag and NDepend as effective tools that can be used in agile development to detect TD in source code. Their findings indicate the need for more tools, models, and guidelines that support the TD management in agile development.

Ampatzoglou et al. (Ampatzoglou et al., 2015) conducted a systematic literature review to understand which are the most common financial terms used in the context of TD management. The collected data produced a glossary and a classification schema of financial approaches used in TD management. They identify seven tools that use this kind of strategy to support TD management. We examined five of them in our study. The search protocol applied in this study did not capture only two tools: AIP and Microsoft Mapper Tree. We excluded these tools because their papers focused on static code analysis, but do not make explicit mention to any technical debt concept.

Li et al. (Li et al., 2015) conducted a systematic mapping study to provide an overview of the current state of research on technical debt management (TDM), including related activities, approaches, and tools. They pointed out a list of 10 TD types, 8 TD management activities, and 29 tools for TD management extracted from 94 primary studies. Regarding technical debt tools, they report their functionality, vendor, TD types and artifacts covered. The research indicates that there is a demand for more dedicated TD management tools. They identified that only 4 tools out of 29 tools are dedicated to TD management. The other 25 tools are adapted for TD identification from other software development areas such as static analysis tools or code smell detection tools. One similarity between their study and ours is the considerable number of analyzed TD tools in both studies. All the 29 tools identified in their study was selected in our systematic mapping study.

Avgeriou et al. (Avgeriou et al., 2020) present an

overview of the current landscape of TD tools, focusing on those offering support for measuring technical debt. The scope of their research limits to examine code, design and architectural TD, comparing them based on the features offered, popularity, empirical validation, and current shortcomings. Our study differs from the Avgeriou et al.'s study because we focus on different kinds of TD tools, not only the ones that provide the TD measurement. Therefore, our study has a broader scope and provides a mappping of state of the art of existing TD tools. Their study focused on a set of 9 tools: CAST, Sonargraph, NDepend, SonarQube, DV8, Squore, CodeMRI, Code Inspector, and SymfonyInsight. The search protocol performed in our study caught the first 6 tools. The last three tools are less popular and have few research work discussing them (Avgeriou et al., 2020).

None of the mentioned studies aims: (i) to characterize the existing TD tools; (ii) to cover the different contexts and activities of software development in which they are applied; and (iii) to investigate the different TD types supported by them. The goal of our work, therefore, differs from the other existing secondary studies in terms of the broad scope of TD tools.

6 CONCLUSIONS

In this paper, we performed a systematic mapping study of 47 primary studies and produced an overview of the state of the art of TD tools. The presented results indicate an increasing interest from the community about technical debt and how to manage it properly in a automated way. All the additional details about our study can be found in (td-tools study, 2020).

We identified a total of 50 tools that supports different TD types and activities. Comparing with previous studies, it represents a huge increasing on the number of TD tools that have been proposed over the last years. 80% (40/50) of these tools provide explicit support to the technical debt concept. Most of tools offer support to managing TD related to the system code, design and/or architecture. This confirms the leading role that code is playing in the field. On the other hand, there is a growing number of TD tools that deals with artifacts other than source code, such as models and requirements. We believe it is fruitful to design and develop tools dealing with artifacts other than source code to manage TD.

A total of 56% (28/50) of the tools are language specific, while the remaining 44% (22/50) of them do not focus in a specific language. Java is the most common target language with 58%. We also found that 84% (42/50) of the tools are original solutions and only 16% represent extensions of existing tools. SonarQube is the current tool with the highest number of extensions.

The different TD management activities received different levels of attention by the tools: (i) most of tools have focused on the identification and measurement activities; (ii) on the other hand, the TD prevention and replacement activities are still not supported by many existing tools. However, we observed that the scientific interest in prevention approaches is increasing. We can conjecture that this trend derives from the awareness of the technical debt cost and that prevention can occasionally be cheaper than its repayment. Similarly, we can notice increasing of the support to the prioritization activity. We believe that the popularization of the TD concept in software development might give rise to new trends for the TDM activities addressed by the tools.

As future work, we plan to conduct a study with practitioners and researchers to compare the tools based on concrete TD management activities and tasks. This would complement the current study with information on the usability and usefulness of the tools.

ACKNOLEDGEMENTS

This work is partially supported by INES (www.ines.org.br), CNPq grant 465614/2014-0, CAPES grant 88887.136410/2017-00, and FACEPE grants APQ-0399-1.03/17 and PRONEX APQ/0388-1.03/14.

REFERENCES

Allman, E. (2012). Managing technical debt. *Commun. ACM*, 55(5):50–55.

Alves, N. S. et al. (2016). Identification and management of technical debt: A systematic mapping study. *Information and Software Technology*, 70:100 – 121.

Ampatzoglou, A. et al. (2015). The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, 64:52–73.

Apa, C. et al. (2020). The perception and management of technical debt in software startups. In *Fundamentals of Software Startups: Essential Engineering and Business Aspects*, pages 61–78. Springer International Publishing, Cham.

Avgeriou, P. et al. (2016). Managing technical debt in software engineering (dagstuhl seminar 16162). *Dagstuhl Reports*, 6(4):110–138.

Avgeriou, P. et al. (2020). An overview and comparison of technical debt measurement tools. *IEEE Software*.

Behutiye, W. N. et al. (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.

Foucault, M. et al. (2018). Gamification: a game changer for managing technical debt? A design study. *CoRR*, abs/1802.02693.

Giraldo, F. D. and Osorio, F. (2017). Evaluating quality issues in BPMN models by extending a technical debt software platform. In *International Conference on Conceptual Modeling*, pages 205–215.

Guo, Y. et al. (2016). Exploring the costs of technical debt management — a case study. *Empirical Softw. Engg.*, 21(1):159–182.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering. Technical report, Technical report, Ver. 2.3 EBSE Technical Report. EBSE.

Kouros, P. et al. (2019). JCaliper: search-based technical debt management. In *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, SAC '19, pages 1721–1730. Association for Computing Machinery.

Lenarduzzi, V. et al. (2019). Technical debt prioritization: State of the art. A systematic literature review. *CoRR*, abs/1904.12538.

Li, Z. et al. (2015). A systematic mapping study on technical debt and its management. *Journal of Systems and Software*, 101:193 – 220.

Mamun, M. et al. (2019). Evolution of technical debt: An exploratory study. In *International Workshop on Software Measurement and International Conference on Software Process and Product Measurement (IWSM Mensura)*, volume 2476, pages 87–102.

Martini, A., Bosch, J., and Chaudron, M. (2015). Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study. *Information and Software Technology*, 67:237 – 253.

Pavlič, L. and Hliš, T. (2019). The technical debt management tools comparison. In *Eighth Workshop on Software Quality Analysis, Monitoring, Improvement, and Applications*, pages 10:1—-10:9.

Petersen, K. et al. (2015). Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and Software Technology*, 64:1 – 18.

Riegel, N. and Doerr, J. (2015). A systematic literature review of requirements prioritization criteria. In Fricker, S. A. and Schneider, K., editors, *Requirements Engineering: Foundation for Software Quality*, pages 300–317, Cham. Springer International Publishing.

Rios, N. et al. (2018). A study of factors that lead development teams to incur technical debt in software projects. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*, pages 429–436.

Rios, N. et al. (2018). A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology*, 102:117–145.

Seaman, C. and Guo, Y. (2011). Measuring and monitoring technical debt. In Zelkowitz, M. V., editor, *Advances in Computers*, volume 82 of *Advances in Computers*, pages 25 – 46. Elsevier.

Suryanarayana, G. et al. (2015). Chapter 8 - repaying technical debt in practice. In Suryanarayana, G., Samarthyam, G., and Sharma, T., editors, *Refactoring for Software Design Smells*, pages 203 – 212. Morgan Kaufmann, Boston.

td-tools study (2020). Project title. https://github.com/td-tools-study/td-tools-study.

Verdecchia, R. et al. (2018). Architectural technical debt identification: The research landscape. In *Proceedings of the 2018 International Conference on Technical Debt*, TechDebt '18, page 11–20, New York, NY, USA. Association for Computing Machinery.

Wieringa, R. et al. (2005). Requirements engineering paper classification and evaluation criteria: A proposal and a discussion. *Requir. Eng.*, 11(1):102–107.

Wohlin, C. et al. (2012). *Experimentation in software engineering*. Springer Science & Business Media.

Yli-Huumo, J. et al. (2016). How do software development teams manage technical debt? – an empirical study. *Journal of Systems and Software*, 120:195–218.