Using Stack Overflow to Assess Technical Debt Identification on Software Projects

Eliakim Gama
State University of Ceará (UECE)
Fortaleza, Brazil
eliakim.gama@aluno.uece.br

Sávio Freire
Federal Institute of Ceará (IFCE)
Morada Nova, Brazil
Federal University of Bahia (UFBA)
Salvador, Brazil
savio.freire@ifce.edu.br

Manoel Mendonça
Federal University of Bahia (UFBA)
Salvador, Brazil
manoel.mendonca@ufba.br

Rodrigo O. Spínola
Salvador University (UNIFACS) and
State University of Bahia (UNEB)
Salvador, Brazil
rodrigo.spinola@unifacs.br

Matheus Paixao
University of Fortaleza (UNIFOR)
Fortaleza, Brazil
matheus.paixao@unifor.br

Mariela I. Cortés
State University of Ceará (UECE)
Fortaleza, Brazil
mariela.cortes@uece.br

ABSTRACT

Context. The accumulation of technical debt (TD) items can lead to risks in software projects, such a gradual decrease in product quality, difficulties in their maintenance, and ultimately the cancellation of the project. To mitigate these risks, developers need means to identify TD items, which enable better documentation and improvements in TD management. Recent literature has proposed different indicator-based strategies for TD identification. However, there is limited empirical evidence to support that developers use these indicators to identify TD in practice. In this context, data from Q&A websites, such as Stack Overflow (SO), have been extensively leveraged in recent studies to investigate software engineering practices from a developers' point of view. *Goal.* This paper seeks to investigate, from the point of view of practitioners, how developers commonly identify TD items in their projects. *Method.* We mined, curated, and selected a total of 140 TD-related discussions on SO, from which we performed both quantitative and qualitative analyses. *Results.* We found that SO's practitioners commonly discuss TD identification, revealing 29 different low-level indicators for recognizing TD items on code, infrastructure, architecture, and tests. We grouped low-level indicators based on their themes, producing an aggregated set of 13 distinct high-level indicators. We then classified all low- and high-level indicators into three different categories according to which type of debt each of them is meant to identify. *Conclusions.* We organize the empirical evidence on the low- and high-level indicators and their relationship to types of TD in a conceptual framework, which may assist developers and serve as guidance for future research, shedding new light on TD identification state-of-practice.

CCS CONCEPTS

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

KEYWORDS

Indicators, Technical Debt, Stack Overflow, Mining Software Repositories.

ACM Reference Format:
Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O. Spínola, Matheus Paixao, and Mariela I. Cortés. 2020. Using Stack Overflow to Assess Technical Debt Identification on Software Projects. In *34th Brazilian Symposium on Software Engineering (SBES '20), October 21–23, 2020, Natal, Brazil.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3422392.3422429

1 INTRODUCTION

Technical debt (TD) refers to immature artifacts resulting from design and implementation decisions that, despite achieving short-term goals, may lead to problems in the maintenance and evolution phases of a software project [14]. Even though TD mostly affects the later phases of development, it can be created and spread in all phases of a software project [16]. While incurring TD can be beneficial when time to market is essential, it only represents a good investment insofar as TD items are effectively managed [9, 14]. TD management involves two main activities: (1) identification and (2) decision making for removal (or not) of TD items, based on the associated economic consequences [4].

Through these activities, software teams become aware of TD items and their long term consequences.

TD identification has attracted attention both in the academic literature [15] and in practitioners' discussion forums [8]. Academic work mostly focuses on looking for TD items in software artifacts according to their quality characteristics [15]. From those characteristics, a set of indicators have been defined to support software teams in recognizing TD items [16]. However, there is limited empirical evidence to support that developers use these indicators to identify TD in their day-to-day practices.

Q&A platforms are paramount in contemporary software development. One of its most prominent examples, Stack Overflow (SO) has emerged as a popular repository of developer knowledge in software engineering. Several studies have used SO data to analyse discussions related to specific topics, such as mobile development [11], web development [17], code smells and anti-patterns [21], microservices [3], and TD [8]. In the work on TD [8], the authors do not focus on any specific TD-related activity. Instead, they only provide a preliminary study focused on understanding the big picture regarding any type of TD discussion on SO. Hence, to the best of our knowledge, no work has investigated to what extent SO's practitioners discuss TD identification and indicators. By leveraging data on SO discussions regarding TD identification, we can provide evidence on how developers identify TD in practice, which can be further used for expanding, confirming and possibly confronting the current empirical knowledge on TD identification [15].

Thus, this work aims at investigating, from the point of view of practitioners on SO, how developers commonly identify TD items in their projects. In particular, we focus on which indicators developers employ to identify TD. To achieve this, we mined, curated, and selected a set of 140 discussions from SO, each of which we further analyzed through qualitative and quantitative procedures. For each discussion, we first identified the type of debt being addressed and the low-level indicators used in its identification. Next, through qualitative analysis and thematic coding [18, 20], we grouped the low-level indicators into high-level indicators.

Results show that TD identification is commonly discussed by SO's practitioners, where they mainly address TD items related to code, infrastructure, architecture, and test debt. We found that the most common indicators discussed by developers on SO are related to *versioning*, *poor coding*, and *comprehensibility* issues. Additionally, we observed that most of the low- and high-level TD indicators discussed on SO are related to code artifacts.

We organized low- and high-level indicators according to their relationships with different types of TD, assembling a conceptual framework for TD identification. This framework presents implications for both practitioners and researchers. While practitioners can employ our proposed framework in the identification of TD in their projects, researchers may view the framework as guidance and inspiration for new research efforts regarding TD identification.

2 TECHNICAL DEBT AND ITS IDENTIFICATION

TD items accumulated over time can reach dire conditions where large management and development efforts are required to repay the debt. This led organizations to employ a more systemic and continuous approach of TD management [4]. However, TD items involve, for the most part, elements that are not directly perceived in commonly analyzed artifacts, such as source code and test cases. These TD items can be related to the system's architecture, documentation and even technological advances [14].

TD management is composed of two main activities. The first is TD identification, which is composed of strategies and approaches for making TD items visible. The second is decision making, which is concerned with the removal (or not) of TD items, based on the associated economic consequences [4]. Several identification techniques have been proposed or adapted, based on the different types of TD from the literature [15]. In general, TD identification involves strategies for assessing software quality characteristics, such as maintainability, good programming and design practices. Strategies are commonly based on some type of indicator, whether formal (e.g., metrics) or not, in order to enable the identification of TD items in different software artifacts [16].

A literature review of TD indicators has been reported by Rios *et al.* [16]. The study identified that a total of 36 indicators have been proposed, discussed and evaluated in the academic literature. When grouped by type of debt, the authors found that *code smells* are the most mentioned TD indicators in the literature. Nevertheless, the empirical evidence that developers employ the TD indicators surveyed in academic literature is still scarce. A better understanding of the indicators developers employ in their TD identification practices may enable better TD monitoring, which in turn can better support TD management and definition of the most suited TD prevention strategies.

3 RESEARCH METHOD

Research Questions

To achieve our research goal, we ask the following research question (RQ) "***How do Stack Overflow's practitioners identify technical debt items in their projects?***". To investigate this question, we propose the following sub-questions:

- ***RQ1:*** *To what extent have Stack Overflow's practitioners discussed the identification of technical debt items?* The purpose of this question is to verify whether practitioners have discussed TD identification in SO.

- ***RQ2:*** *What types of technical debt are addressed in discussions regarding TD identification in the SO platform?* We intend to identify the types of debt that are discussed by practitioners on SO.
- ***RQ3:*** *What are the indicators employed by Stack Overflow's practitioners to identify technical debt items?* Through this question, we seek to distinguish the indicators used by SO's practitioners to identify TD items in their projects.
- ***RQ4:*** *What is the relationship between TD indicators discussed by Stack Overflow's practitioners and type of debt?* This question investigates how the indicators are related to types of debt, such as code, architecture, and test debt.

Data Collection

We employed a data collection procedure inspired by the work of Gama *et al.* [8]. We leveraged the *SOTorrent* dataset [2] (version 06/21/19), which is a curated dataset extracted directly from Stack Overflow's data dump. From SOTorrent, one can access data regarding SO's questions, answers, comments, links to other repositories and other metadata.

First, we collected all questions and answers in which we could find the string *"technical debt"*. We applied this string matching procedure in the questions' title, questions' body, questions' tags and answers' body. According to SO's guidelines [19], comments are not considered proper answers to questions. Hence, comments were ignored by this procedure. The string matching phase yielded a total of **773** discussions.

Next, we performed a filtration process, as shown in Figure 1. The filtering consisted of 4 steps, each of which are depicted in the figure and detailed as follows.

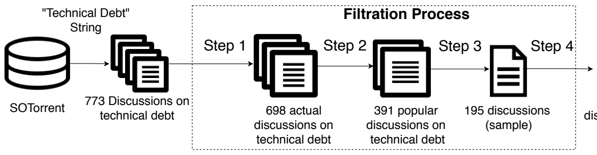
"Technical Debt" String
SOTorrent
773 Discussions on technical debt
Filtration Process
Step 1
Step 2
Step 3
Step 4
698 actual discussions on technical debt
391 popular discussions on technical debt
195 discussions (sample)
140 discussions

Figure 1: Data collection process to extract TD-related discussions on SO

Step 1: We eliminated incomplete discussions from the data set. Hence, we consider a discussion to consist of a question and one (or more) answers, where there is at least one answer whose author is not the author of the question [3, 8]. After applying this criterion, **698** discussions remained out of the initial **773**.

Step 2: As data from SO is known to be affected by noise [1, 10], we mitigated it applying the following different proxies for popularity: number of answers, number of views, number of comments, number of favorites, and score (internal SO's popularity metric). For each of these metrics, we ranked the discussions and extracted all above the third quartile of the distribution, resulting in a population of **391** discussions.

Step 3: We noticed that the string *"technical debt"* can be found in different parts of the discussion (its question, its answers, or both). As we intended to extract a statistically significant sample of the population with 95% confidence and 5% error rate, we performed a stratified sampling technique, where **18%** were sampled from discussions in which the string appears in the question, **76%** in the answers, and **6%** in both. This procedure resulted in **195** discussions.

Step 4: After an initial analysis of the selected discussions, we noticed that some of them were false positives, i.e., a discussion containing the string *"technical debt"* where the topics being addressed were not related to technical debt[1]. To rule-out the false positives from our sample, we conducted a manual qualitative analysis in the discussions, in which all discussions were separately analyzed by two authors and divergences were revolved by those authors and other ones. After false positives removal, our final sample of TD-related discussions on SO was composed of **140** discussions.

Data Analysis

For each of the 140 discussions in our sample, we performed a manual qualitative analysis to identify the TD indicators and the type of debt being addressed by SO's practitioners in the discussion. The analysis consisted of two authors independently reading, analyzing and coding each discussion, searching for TD indicators and type of debt being addressed. In case of divergence between authors, the discussion was further analyzed by a third author to reach a consensus.

To identify the types of debt addressed in the discussions, we employed the definitions reported by Rios *et al.* [15]. For instance, one of the discussions in our sample goes as follows[2]: "*(...). Also, even if your system is small at the moment, you might want to add more features to it later – not going N-tiered might (sic) constitute a sort of technical debt on your part, so you have to be careful.*". This discussion occurs in the context of problems that might happen due to architectural evolution, characterizing a discussion regarding *architecture debt.*

The process followed for the identification of TD indicators in the discussions was based on qualitative data analysis and manual coding [18, 20]. For each discussion, we tagged relevant pieces of text (regarding TD indicators) with codes. Codes with the same meaning were unified in **low-level indicators**. This process was performed incrementally until no new codes were identified (point of saturation). The complete manual analysis and coding required some effort. Consider the following quotes found in three discussions: *"messy code"*, *"codebases maelstrom"*, and *"team that does not*

[1]https://stackoverflow.com/questions/21867892/project-coverage-is-set-to-0-jacoco-and-sonar-in-eclipse
[2]https://stackoverflow.com/questions/5880/are-there-any-negative-reasons-to-use-an-n-tier-solution

understand its code", respectively. These codes are related to difficulties in understanding code. Hence, they are coded as the low-level indicator *low code comprehensibility.*

Since many low-level indicators were related to each other, we followed a grouping procedure [18, 20] to organize them into **high-level indicators**. A joint effort of this paper's authors analyzed the low-level indicators, found relationships between them, and grouped them into high-level indicators. To name the high-level indicators, we used a broad generic name that represents each group. For instance, the low-level indicators *lack of unit testing*, *lack of test coverage*, and *time delay in performing tests* were grouped into the high-level indicator *poor testing practices.*

We followed the same grouping process to structure the high-level indicators into **categories**, according to the ones defined by Freire *et al.* [6]. For instance, we used the category *infrastructure* to group high-level indicators such as *issues with third-party software* and *presence of database issues.*

Lastly, we associated a certain low-level indicator with a certain type of debt when both are found in the same discussion. In addition, we computed the frequency of occurrence of a high-level indicator regarded to a certain TD type as the frequency of occurrence of the respective low-level indicators regarded to the TD type.

To allow for the reproducibility of our work, we make available a replication package containing the complete set of raw data, analyses, and results of this paper [7].

4 RESULTS

To What Extent Have SO's practitioners discussed the identification of technical debt items? *(RQ1)*

Initially, we verified whether SO's practitioners have addressed TD identification in their discussions. Out of the 140 discussions under analysis, SO practitioners have addressed TD identification in 99 (~71%) of them. Next, we provide an example of discussion in our sample that addresses TD identification[3]: *"I have worked on technical debts and found the below issue, maybe false positive for the below C++ code: (...)".*

What types of technical debt are addressed in discussions regarding TD identification in the SO platform? *(RQ2)*

When looking at the discussions that address TD identification, we managed to find six different types of debt being discussed. Note that some discussions may have addressed more than one type of debt. Table 1 reports on the types of debt being discussed, followed by the number of times (in all discussions) that a type of debt has been mentioned (**#TDT**), and the percentage in relation to the total number of types mentioned overall (**%TDT**). We can observe that the majority of discussions regarding TD identification are related to *code debt*, followed by *infrastructure*, *architecture*, and *test debt.*

When comparing to the list of types of debt found by Rios *et al.* [15], practitioners from SO have not discussed indicators to identify TD items of design, documentation, requirements, people, build, process, automation test, service, and versioning debt.

Table 1: Types of debt addressed in discussions regarding TD identification on the SO platform

Type of debt	#TDT	%TDT
Code debt	52	50.0%
Infrastructure debt	18	17.3%
Architecture debt	16	15.4%
Test debt	16	15.4%
Defect debt	1	1.0%
Usability debt	1	1.0%

What are the indicators employed by SO's practitioners to identify technical debt items? *(RQ3)*

As a result of the coding process, we identified **29 low-level indicators** used by SO's practitioners for recognizing TD items in their projects. The complete set of low-level indicators are depicted in Figure 3. We noticed that the top 10 low-level indicators, presented in Table 2, correspond to ~60% of the overall number of mentions. In addition, we present in the table the number of mentions of each low-level indicator (**#LLI**), and the percentage in relation to the total mentions of low-level indicators (**%LLI**). We can observe that *technology version issues*, *bad coding*, *low code comprehensibility*, and *database issues* are the most mentioned low-level indicators found in SO discussions regarding TD identification.

Table 2: Top 10 low-level indicators for the identification of TD items, as discussed by SO's practitioners

Low-level indicator of TD items	#LLI	%LLI
Technology version issues	12	11.54%
Bad coding	9	8.65%
Low code comprehensibility	8	7.69%
Database issues	7	6.73%
Use of workarounds	6	5.77%
Lack of test coverage	5	4.81%
Code smell	4	3.85%
Duplicate code	4	3.85%
Lack of testing	4	3.85%
Dead code	3	2.88%

Considering the top 10 low-level indicators presented on the table, we can notice that some of them are closely related.

[3]https://stackoverflow.com/questions/48495165/string-returned-from-function-refer-to-local-object-address

For instance, *low code comprehensibility*, *bad coding*, *code smell*, *duplicate code*, *dead code*, and *use of workarounds* are associated with problems in code or design. The low-level indicators *technology version issues* and *database issues* are related to infrastructure issues. Finally, the low-level indicators *lack of test coverage* and *lack of testing* are related to poor testing practices.

Hence, based on the relationship between the 29 low-level indicators, we grouped them into **13 high-level indicators**. For example, low-level indicators *code smells*, *duplicate code*, and *dead code* indicate characteristic in the source code and design that could become a problem [5, 13]. Thus, these low-level indicators were grouped into the high-level indicator *presence of code smells* (Figure 3).

Table 3 presents the top 5 high-level indicators, regarding the number of times that were mentioned in discussions by SO's practitioners. In addition, for each high-level indicator, we report the number of associated low-level indicators (**#LLI**), the total of (accumulated) mentions for the high-level indicator (**#HLI**), and the percentage of mentions in relation to all mentioned high-level indicators (**%HLI**).

Table 3: Top 5 high-level indicators for identifying TD items

High-level indicator for TD items	#LLI	#HLI	%HLI
Presence of bad coding	3	20	19.23%
Poor testing practices	5	15	14.42%
Technology version and update issues	2	13	12.50%
Lack of good design practices	6	12	11.54%
Presence of code smells	3	11	10.58%

We can notice that the high-level indicator *lack of good design practices* concentrated the largest number of low-level indicators, indicating that this indicator can be employed in different situations to identify TD items. The high-level indicator *poor testing practices* concentrated five low-level indicators while the others concentrated at least two low-level indicators. Finally, as we can see from Table 3, the top 5 high-level indicators corresponds to ~68% of the overall number of mentions.

We detail each of the top high-level indicators as follows:

- **Presence of bad coding** is related to issues presented in source code. This indicator is composed of the following low-level indicators: *bad coding*, *lack of code quality*, and *low code comprehensibility*.
- **Poor testing practices** is related to the lack or poor practices regarding software testing. This indicator is composed of low-level indicators such as: *lack of test coverage*, *lack of unit testing*, and *time delay in performing tests*.
- **Technology version and update issues** is associated with problems in versioning or updating of technologies. *Technology version issues* and *framework updates issues* are the low-level indicators within this high-level indicator.
- **Lack of good design practices** is associated with bad practices adopted by the team regarding the software's design. *Lack of design patterns*, *no adoption of development patterns*, and *no adoption of naming conventions* are some of the low-level indicators within this high-level indicator.
- **Presence of code smells** is related to characteristics in the source code or design that may indicate a deeper problem. This high-level indicator is composed of low-level indicators *code smells*, *duplicate* and *dead code*.

The list of all high-level indicators, their description, and quotes supporting their identification in SO discussions are available at our replication package [7].

Grouping high-level TD indicators into categories. By grouping all 13 high-level indicators, we identified the following three categories:

- **Development Issues**: high-level indicators related to software development activities. Among them, we have *presence of bad code*, *lack of good design practices*, and *presence of code smells*.
- **Infrastructure**: high-level indicators that encompass issues related to tools, technologies, and development environments. Among them, we have *technology version and update issues*, *presence of database issues*, and *legacy architecture and code*.
- **Methodology**: high-level indicators associated with processes and activities. The only high-level indicator in this is *poor testing practices*.

Table 4 shows the identified categories, reporting the number of high-level indicators composing the category (**#HLI**) and the number of mentions of the category in the analyzed discussions (**#CM**). Column **%CM** indicates the percentage of #CM in relation to all mentions we found. We can notice that the most mentioned category (*development issues*) represents ~55% of the total number of mentions, playing a central role in TD identification for SO's practitioner. In addition, this category concentrates the most number of high-level indicators. The category *infrastructure* is responsible for ~31% of the overall mentions and is composed of five high-level indicators, while the category *methodology* has ~13% of the total of mentions with one high-level indicator.

Table 4: Categories of high-level TD indicators

Category	#HLI	#CM	%CM
Development Issues	7	57	54.81%
Infrastructure	5	32	30.77%
Methodology	1	15	14.42%

Figure 2 depicts a mind map of the categories and their high-level indicators. The number in parentheses represents the percentage of mentions of a category or high-level indicator related to the total of mentioned categories or the total number of high-level indicators mentioned in the same category, respectively. We can observe that *presence of bad coding* and *lack of good design practices* correspond to ~19% of the overall mentions in the *development issues* category. In the *infrastructure* category, the most mentioned high-level indicators are *technology version and update issues* and *presence of database issues* with ~13% of the mentions in the category. Lastly, the high-level indicator *poor testing practices* corresponds to ~14% in the *methodology* category.

What is the relationship between TD indicators discussed by Stack Overflow's practitioners and type of debt? *(RQ4)*

To answer RQ4, we performed a two-step data analysis. First, we consider the relationship between types of debt and related low-level indicators, as previously presented. Secondly, we mapped the relationship between types of debt and high-level indicators by considering the relationships between types and low-level indicators defined in the previous step.

Table 5 shows the relationship between types of debt and the top 10 low-level indicators, indicating the low-level indicator name, the type of debt, and the number of occurrences among them. We used acronyms to identify each type of debt, such as **CD** for code debt, **ID** for infrastructure debt, **AD** for architecture debt, **TD** for test debt, **DD** for design debt, and **UD** for usability debt. We can notice that items of *code debt* can be recognized by almost all top low-level indicators (except *lack of testing*), while items of *defect debt* cannot be identified by any of the top low-level indicators. Apart from the low-level indicators *bad coding*, *duplicate code*, *lack of testing*, and *dead code*, all others have been discussed to identify TD items with more than two types of debt.

Table 5: Relationship between the top 10 low-level TD indicators and types of debt

Low-level indicator	CD	ID	AD	TD	DD	UD
Technology version issues	5	6	0	1	0	0
Bad coding	8	1	0	0	0	0
Low code comprehensibility	6	0	1	1	0	0
Database issues	4	2	1	0	0	0
Use of workarounds	3	0	2	0	0	1
Lack of test coverage	1	0	1	3	0	0
Code smell	2	1	1	0	0	0
Duplicate code	3	0	0	1	0	0
Lack of testing	0	0	1	3	0	0
Dead Code	3	0	0	0	0	0

Table 6 shows the relationship between types of debt and the top 5 high-level indicators, reporting the high-level indicator name, the type of debt, and the number of occurrences between each other. We used the same acronyms reported in the previous paragraph to identify each type of debt. We can observe that items of *code debt* and *architecture debt* can be identified for almost all high-level indicators. On the other side, items of *defect debt* and *usability debt* are not identified by any of the top high-level indicators. The high-level indicators *presence of bad coding*, *lack of good design practices*, and *presence of code smells* are used to identify TD items of most different types of debt. These indicators can identify the same types of debt: *code debt*, *infrastructure debt*, *architecture debt*, and *test debt*. Those types of debt are the most commonly found in SO's discussions on TD identification.

Table 6: Relationship between the top 5 high-level TD items indicators and types of debt

High-level indicator	CD	ID	AD	TD	DD	UD
Presence of bad coding	16	2	1	1	0	0
Poor testing practices	1	0	2	12	0	0
Technology version and update issues	5	7	0	1	0	0
Lack of good design practices	8	1	2	1	0	0
Presence of code smells	8	1	1	1	0	0

Lastly, Table 7 shows the relationship between types of debt and the categories of TD indicators, reporting the category name, the type of debt, and the number of occurrences between each other. We used the same acronyms reported in the previous paragraph to identify each type of debt. We can notice that items of *code debt*, *architecture debt*, and *test debt* can be identified by all categories of indicators. *Defect* and *usability* debt items can be only recognized by indicators from the *development issues* category. This category can spot items of all types of debt addressed in SO's discussions regarding TD identification.

Table 7: Relationship between Categories of TD Indicator and Types of Debt

Categories of TD Indicator	CD	ID	AD	TD	DD	UD
Development Issues	38	4	10	3	1	1
Infrastructure	13	14	4	1	0	0
Methodology	1	0	2	12	0	0

5 CONCEPTUAL FRAMEWORK FOR TECHNICAL DEBT IDENTIFICATION

Figure 3 shows the *conceptual framework* regarding TD identification we elaborated based on the results and observations reported in this empirical study. We envision the framework

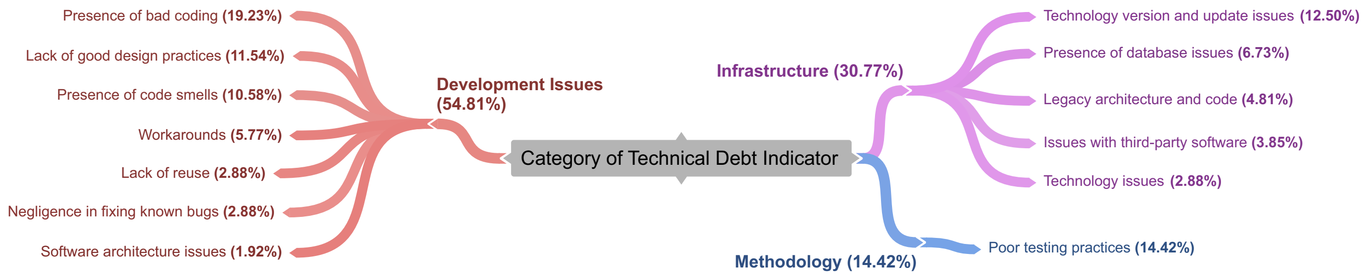
Presence of bad coding (19.23%)
Lack of good design practices (11.54%)
Presence of code smells (10.58%)
Workarounds (5.77%)
Lack of reuse (2.88%)
Negligence in fixing known bugs (2.88%)
Software architecture issues (1.92%)
Development Issues (54.81%)
Category of Technical Debt Indicator
Infrastructure (30.77%)
Methodology (14.42%)
Technology version and update issues (12.50%)
Presence of database issues (6.73%)
Legacy architecture and code (4.81%)
Issues with third-party software (3.85%)
Technology issues (2.88%)
Poor testing practices (14.42%)

Figure 2: Technical Debt Indicator Categories

being used to support development teams in TD identification activities. The framework is inspired by the Sankey diagram, which has been used to visualize flows of energy, materials or other resources in a variety of applications [12]. This diagram is composed of nodes representing the source and links shows the magnitude of flow between nodes.

We chose this diagram because it makes it possible to define the TD identification flow. This flow can be read, interpreted, and used by software teams in two-ways. While writing or inspecting code, a developer might recognize one of the high- or low-level indicators. In this case, the developer will be able to know which type of debt that particular indicator is pointing towards. Differently, if a software team decides to identify the TD items regarding a certain type of debt, the developer can use the associated high- and low-level indicators as starting points.

We adapted the diagram by defining high-level indicators, low-level indicators, and types of debt as nodes. The relationship between them is mapped as links. The flow starts at a high-level indicator and is distributed across its low-level indicators. Thereafter, the flow of a low-level indicator is distributed across types of debt. We defined this flow because high-level indicators are composed of low-level indicators, and they are related to types of debt. In these flow distributions, the magnitude of the link is represented by its width. The greater the magnitude, the wider the link.

To compute a link's magnitude, we used the high- and low-level indicators. First, we summed the number of occurrences of each high- or low-level indicator. We computed the percentages for each high- or low-level indicator according to the total number of occurrences of all high- or low-level indicators. We followed this procedure for computing the percentage for low-level indicators and types of debt.

In Figure 3, we can notice that the high-level indicator *presence of bad coding* represents 20% of the overall mentions of all high-level indicators. By analysing its flow, this indicator is composed of the following low-level indicators: *bad coding*, *low code comprehensibility*, and *lack of code quality*, with magnitude values of 9%, 8%, and 3%, respectively. This demonstrates that the low-level indicator *bad coding* is the most common in its high-level indicator (*presence of bad coding*). Considering the low-level indicator *low code comprehensibility*, we can observe that it spreads out to *code debt*, *architecture debt*, and *test debt* with magnitude of 6%, 1%, and 1%, respectively. This indicates that a TD item identified with this indicator is more likely to be related to code debt, instead of the other types of debt.

6 SUMMARY OF FINDINGS

We present the main findings of this paper as follows:

- The identification of TD items is commonly the subject of TD-related discussions on Stack Overflow;
- Code and infrastructure debt are the most discussed types of debt when SO's practitioners address TD identification;
- *Technology version issues*, *bad coding*, and *low code comprehensibility* are the most mentioned low-level indicators for TD identification;
- *Presence of bad coding*, *poor testing practices*, and *technology version and update issues* are the most mentioned high-level indicators for TD identification.
- TD indicators are more concentrated in the *development issues* category;
- We created a conceptual framework regarding TD identification involving all high- and low-level indicators, and their relationship with different types of debt.

7 RELATED WORK

Previous work have analysed SO data for different technical topics, such as mobile development [11], web development [17], code smells and anti-patterns [21], and microservices [3]. Regarding TD, Gama *et al.* [8] performed a preliminary study of TD in the SO's platform, evidencing that TD is discussed by SO' practitioners. On the other side, Rios *et al.* [15] reported a list of situations where it is possible to find TD items, supporting the definition of each type of debt. In a previous work, Rios *et al.* [16] conducted a mapping study on TD and its management, revealing a set of indicators divided by types of debt that can be used for identifying TD items. Since this work is the most similar to ours, we performed a comparison of results, as detailed next.

To perform the comparison, the first and second authors of this paper analyzed and mapped the indicators defined in

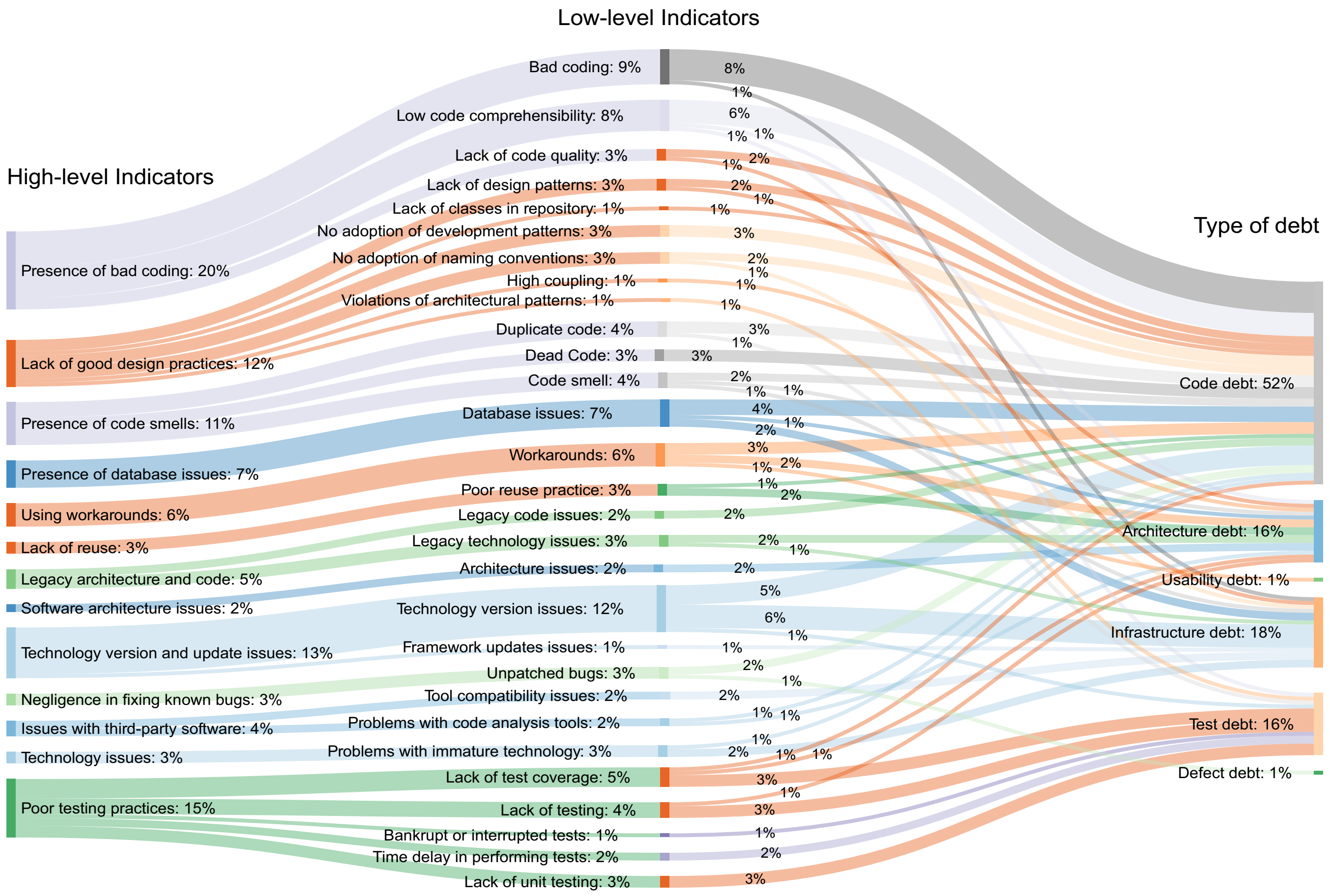
Low-level Indicators
High-level Indicators
Type of debt
Presence of bad coding: 20%
Lack of good design practices: 12%
Presence of code smells: 11%
Presence of database issues: 7%
Using workarounds: 6%
Lack of reuse: 3%
Legacy architecture and code: 5%
Software architecture issues: 2%
Technology version and update issues: 13%
Negligence in fixing known bugs: 3%
Issues with third-party software: 4%
Technology issues: 3%
Poor testing practices: 15%
Bad coding: 9%
Low code comprehensibility: 8%
Lack of code quality: 3%
Lack of design patterns: 3%
Lack of classes in repository: 1%
No adoption of development patterns: 3%
No adoption of naming conventions: 3%
High coupling: 1%
Violations of architectural patterns: 1%
Duplicate code: 4%
Dead Code: 3%
Code smell: 4%
Database issues: 7%
Workarounds: 6%
Poor reuse practice: 3%
Legacy code issues: 2%
Legacy technology issues: 3%
Architecture issues: 2%
Technology version issues: 12%
Framework updates issues: 1%
Unpatched bugs: 3%
Tool compatibility issues: 2%
Problems with code analysis tools: 2%
Problems with immature technology: 3%
Lack of test coverage: 5%
Lack of testing: 4%
Bankrupt or interrupted tests: 1%
Time delay in performing tests: 2%
Lack of unit testing: 3%
Code debt: 52%
Architecture debt: 16%
Usability debt: 1%
Infrastructure debt: 18%
Test debt: 16%
Defect debt: 1%

Figure 3: Conceptual Framework for Technical Debt Identification

our study with the ones found by Rios *et al.* [16], separately. As those indicators have different levels of abstraction, the authors performed the comparison using the list of low-level indicators. Subsequently, the results were consolidated considering the respective high-level indicator for each low-level one. The divergences were resolved in a consensus meeting involving other authors.

Table 8 presents the (high-level) indicators that were equivalent to the ones reported in our paper and the ones found by Rios *et al.* [16]. Besides, the table reports the types of debt associated with each high-level indicator. The results of this comparison are available at our replication package [7].

Except for the high-level indicators *legacy architecture and code* and *technology issues*, all others have equivalence to an indicator found by Rios *et al.* [16]. We noticed that some of our high-level indicators are equivalent to more than one indicators reported by Rios. For instance, the indicator *lack of good design practices* is associated with *software architecture issues*, *code without standards*, *code metrics (not specified)*, *software design issues*, and *afferent / efferent couplings (AC/EC)*.

Regarding types of debt, we observed that some equivalent indicators do not share the same types of debt. For example, the high-level indicator *poor testing practices* is associated with architecture, code, and test debt in our conceptual framework while its correspondent (*lack of automated testing*) is only related to test automation debt in the related work. Unlike the indicators of Rios *et al.* [16], our study did not found high-level indicators associated with design debt. It happens because SO's practitioners discussed code and architecture aspects more than design ones.

In summary, our findings confirm the indicators found by Rios *et al.* [16], and complement those adding more empirical evidence about the relationship between TD indicators and types of debt.

8 IMPLICATIONS FOR PRACTITIONERS AND RESEARCHERS

Practitioners may employ our conceptual framework in the identification of TD items. Since the framework presents a set of indicators and their relationships with types of debt, it can be used as a starting point in meetings for identifying

Table 8: Comparison to Related Work

Our Study		Study of Rios *et al.* [16]	
Type of debt	**High-level indicator**	**Indicator**	**Type of debt**
Architecture debt, code debt, and infrastructure debt	Presence of database issues	Referential integrity constraints (RICs)	Design debt
Architecture debt, code debt, infrastructure debt, and test debt	Lack of good design practices	Code metrics (not specified)	Design debt and code debt
		Code without standards	Code debt
		Software architecture issues	Architecture debt
		Software design issues	Design debt
		Afferent and efferent couplings (AC / EC)	Design debt
Code debt and defect debt	Negligence in fixing known bugs	Uncorrected known defects	Defect debt and test debt
		Defects deferred	Test debt
Architecture debt, code debt, and test debt	Poor testing practices	Incomplete tests	Test debt
		Insufficient code coverage	Test debt
		Lack of automated testing	Test automation debt
Architecture debt, code debt, infrastructure debt, and test debt	Presence of bad coding	Code without standards	Code debt
		Low external / internal quality	Design debt
Architecture debt, code debt, test debt, and infrastructure debt	Presence of code smells	Code smells	Design debt and code debt
Architecture debt and code debt	Issues with third-party software	Automatic static analysis (ASA) issues	Design debt and code debt
Architecture debt	Software architecture issues	Software architecture issues	Architecture debt
Infrastructure debt	Technology version and update issues	Software architecture issues	Architecture debt
Architecture debt and code debt	Lack of reuse	Software design issues	Design debt
Architecture debt, code debt, and usability debt	Workarounds	Low external and/or internal quality	Design debt

TD items. For example, when a software team recognizes a problem in its project, the team can verify whether this problem is a TD indicator, according to the framework. Next, the team can visualize the possible types of debt they are facing, and define (i) payment actions to eliminate them or (ii) preventive actions to avoid them in the future. In addition, the framework can be used as a complement for developers' existing knowledge regarding TD items and their indicators.

For researchers, our findings support new research efforts in TD identification. Our list of low-level indicators can guide new investigations on metrics and their thresholds to specialize the low-level indicators for different software teams. The relationship between high-level indicators and types of debt demonstrates the possibility of how types of debt may be related to each other.

9 THREATS TO VALIDITY

This section discusses the threats to the validity of our study, and the mitigation actions we have taken. The threats are detailed following the classification defined in [22].
Construct: a threat arises from the process of choosing the discussions from SO, as we did not survey SO's practitioners. To mitigate this threat, we applied a selection methodology that has been employed in recent empirical studies regarding SO data [3, 8]. In addition, we performed a filtration process, ensuring the minimization of noise in our analyses.
Internal: the processes used for analyzing and coding the discussions, grouping low-level indicators into high-level indicators, and grouping high-level indicators in categories can represent a threat in our study. To avoid this, the aforementioned procedures were performed by two researchers, separately. Besides, the divergences were resolved in a consensus meeting between those researchers and at least one more researcher.
External: Although the study provides representative results on TD identification, the results cannot yet be generalized. We reduce this threat by considering a representative sample of discussions from SO, the largest Q&A platform focused on computer science discussions nowadays. An extension of this study that considers the entire population of TD discussions on SO, and not only a sample, would possibly yield more generalizable results.
Conclusion: The subjectivity involved in the analysis and interpretation of the discussions, and the coding process might lead researchers to different conclusions, even having adopted the same analysis method. We sought to reduce this threat by thoroughly following, and explaining, our analysis process(Section 3). Moreover, we make all our results, analyses, and observations available in our replication package [7].

10 FINAL REMARKS

This work aims at investigating, from the point of view of practitioners on SO, how developers commonly identify TD items in their projects. We noticed that strategies and approaches for identifying TD items are addressed in discussions related to TD. Based on a manual coding process, we report the most common high- and low-level indicators SO's practitioners have discussed for recognizing TD items in their projects. Also, we associate each of these indicators with types of debt. Finally, we leverage the observations regarding all high- and low-level TD indicators and their relationship to different types of debt to propose a conceptual framework for TD identification. Our conceptual framework can be used by software teams to facilitate the identification of TD items in their projects.

The future directions for this research include: (i) to analyze the complete population of discussions instead of a sample to increase the external validity of our results, and (ii) to empirically assess the proposed conceptual framework regarding its effectiveness in supporting TD identification in real-world software systems. Moreover, we envision the study of SO discussions regarding other activities in the TD management cycle, such as TD payment and prevention.

ACKNOWLEDGMENTS

This study was financed in part by the Coordenação de Aperfeiçoamento de Pessoal de Nível Superior Brasil (CAPES) Finance Code 001. This research was also supported in part by funds received from the David A. Wilson Award for Excellence in Teaching and Learning, which was created by the Laureate International Universities network to support research focused on teaching and learning. For more information on the award or Laureate, please visit www.laureate.net.

REFERENCES

[1] Muhammad Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K. Roy, and Kevin A. Schneider. 2016. Mining duplicate questions in stack overflow. In *Proceedings of the 13th International Workshop on Mining Software Repositories*. ACM Press, New York, New York, USA, 402–412.

[2] Sebastian Baltes, Christoph Treude, and Stephan Diehl. 2019. Sotorrent: Studying the origin, evolution, and usage of stack overflow code snippets. In *Proceedings of the 16th International Conference on Mining Software Repositories*. IEEE Press.

[3] Alan Bandeira, Carlos Alberto Medeiros, Matheus Paixao, and Paulo Henrique Maia. 2019. We Need to Talk about Microservices: an Analysis from the Discussions on StackOverflow. *International Conference on Mining Software Repositories* 5 (2019), 255–259.

[4] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, and M. S. Keymind. 2013. Practical considerations, challenges, and requirements of tool-support for managing technical debt, Vol. 00. 16–19.

[5] Martin Fowler, Kent Beck, John Brant, William Opdyke, and don Roberts. 2002. *Refactoring: Improving the Design of Existing Code*. PEARSON EDUCATION.

[6] Sávio Freire, Nicolli Rios, Boris Gutierrez, Darío Torres, Manoel Mendonça, Clemente Izurieta, Carolyn Seaman, and Rodrigo O. Spínola. 2020. Surveying Software Practitioners on Technical Debt Payment Practices and Reasons for Not Paying off Debt Items. In *Proceedings of the Evaluation and Assessment in Software Engineering (EASE '20)*. ACM, New York, NY, USA, 210–219. https://doi.org/10.1145/3383219.3383241

[7] Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O. Spínola, Matheus Paixao, and Mariela I. Cortés. 2020. Replication package for the paper: "Using Stack Overflow to Assess Technical Debt Identification on Software Projects". https://zenodo.org/record/3998025

[8] Eliakim Gama, Matheus Paixao, Emmanuel Sávio Silva Freire, and Mariela Inés Cortés. 2019. Technical Debt's State of Practice on Stack Overflow. In *Proceedings of the XVIII Brazilian Symposium on Software Quality - SBQS'19*. ACM Press, New York, New York, USA, 228–233.

[9] Yuepu Guo, Rodrigo Oliveira Spínola, and Carolyn Seaman. 2016. Exploring the Costs of Technical Debt Management — a Case Study. *Empirical Softw. Engg.* 21, 1 (2016), 159–182.

[10] David Kavaler, Daryl Posnett, Clint Gibler, Hao Chen, Premkumar Devanbu, and Vladimir Filkov. 2013. Using and asking: Apis used in the android market and asked about in stackoverflow. In *International Conference on Social Informatics*. Springer, 405–418.

[11] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. How Do API Changes Trigger Stack Overflow Discussions? A Study on the Android SDK. In *Proceedings of the 22nd International Conference on Program Comprehension* (Hyderabad, India) *(ICPC 2014)*. Association for Computing Machinery, New York, NY, USA, 83–94. https://doi.org/10.1145/2597008.2597155

[12] R.C. Lupton and J.M. Allwood. 2017. Hybrid Sankey diagrams: Visual analysis of multidimensional data for understanding resource use. *Resources, Conservation and Recycling* 124 (2017), 141 – 151. https://doi.org/10.1016/j.resconrec.2017.05.002

[13] Mika Mäntylä and Casper Lassenius. 2006. Subjective evaluation of software evolvability using code smells: An empirical study. *Empirical Software Engineering* 11, 3 (2006), 395–431. http://dblp.uni-trier.de/db/journals/ese/ese11.html#MantylaL06

[14] I. Ozkaya, R. L. Nord, and P. Kruchten. 2012. Technical Debt: From Metaphor to Theory and Practice. *IEEE Software* 29 (2012), 18–21.

[15] Nicolli Rios, Manoel Mendonça, and Rodrigo O. Spínola. 2018. A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. *Information and Software Technology* 102 (2018), 117 – 145.

[16] Nicolli Rios, Thiago S.Mendes, Manoel Mendonça, Rodrigo O. Spínola, Forrest Shull, and Carolyn Seaman. 2016. Identification and management of technical debt: A systematic mapping study. 70 (2016), 100 – 121.

[17] Christoffer Rosen and Emad Shihab. 2016. What Are Mobile Developers Asking about? A Large Scale Study Using Stack Overflow. *Empirical Softw. Engg.* 21, 3 (June 2016), 1192–1223. https://doi.org/10.1007/s10664-015-9379-3

[18] Carolyn B. Seaman. 1999. Qualitative Methods in Empirical Studies of Software Engineering. *IEEE Trans. Softw. Eng.* 25, 4 (July 1999), 557–572. https://doi.org/10.1109/32.799955

[19] StackOverflow. 2020. Stack Overflow Annual Developer Survey. https://stackoverflow.com/help/privileges Accessed: 2020-06-08.

[20] Anselm L. Strauss and Juliet M. Corbin. 1998. *Basics of qualitative research: techniques and procedures for developing grounded theory*. Sage Publications, Thousand Oaks, Calif. XIII, 312 s pages.

[21] Amjed Tahir, Aiko Yamashita, Sherlock Licorish, Jens Dietrich, and Steve Counsell. 2018. Can You Tell Me If It Smells? A Study on How Developers Discuss Code Smells and Anti-Patterns in Stack Overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018* (Christchurch, New Zealand) *(EASE'18)*. Association for Computing Machinery, New York, NY, USA, 68–78. https://doi.org/10.1145/3210459.3210466

[22] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.