Understanding Practitioners' Reasoning and Requirements for Efficient Tool Support in Technical Debt Management

João Paulo Biazotto[a, b*], Daniel Feitosa[a], Paris Avgeriou[a], Elisa Yumi Nakagawa[b]

[a]Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands.
[b]Institute of Mathematics and Computational Sciences, University of São Paulo, Brazil.

*Corresponding author(s). E-mail(s): j.p.biazotto@rug.nl;
Contributing authors: d.feitosa@rug.nl; p.avgeriou@rug.nl; elisa@icmc.usp.br;

Abstract

Context: Maintaining software projects over the long term requires controlling the accumulation of technical debt (TD). However, the time and cost associated with technical debt management (TDM) are often high, hindering practitioners from performing TDM tasks. Using tools for TDM has the potential to reduce the effort involved. Despite this, the adoption of such tools remains low, indicating a need for more efficient tool support.
Objective: This study aims to understand practitioners' perspectives on tool support for TDM, specifically regarding their selection and use of these tools. Additionally, we seek to identify potential requirements that could be implemented into existing or new TDM tools.
Method: We surveyed practitioners and received 103 answers, from which 89 valid answers were analyzed using both thematic synthesis and descriptive statistics.
Results: Practitioners' decision-making processes regarding adopting tools are primarily driven by ten main concerns identified from practitioners' responses (e.g., the load of information provided by tools). Additionally, we elicited 46 requirements and classified them into two main categories (*"Information to be provided"* and *"Tool Usage"*).

Conclusion: Practitioners aim to maintain control over tool execution and outputs. Therefore, our study highlights the necessity of human-centered approaches for TDM automation, i.e., not only tools are essential, but the interaction between tools and practitioners is critical for a more efficient TDM.

Keywords: technical debt; technical debt management; tool support; survey; thematic synthesis;

1 Introduction

The term technical debt (TD) was proposed in the 1990s to describe non-optimal decisions that software developers make (Cunningham, 1992). Such decisions can be made due to a variety of reasons, such as insufficient resources, lack of knowledge, or tight deadlines (Rios et al., 2018). A few examples of TD items are code smells (Sas et al., 2021), requirements only partially implemented (Melo et al., 2022; Rios et al., 2018), and low-quality code (Digkas et al., 2022). When incurring TD, practitioners usually optimize short-term benefits, such as improving the speed of development or reducing its cost (Rios et al., 2020). However, in the long term, the accumulation of TD can harm the software's maintainability and evolvability (Li et al., 2015). Hence, it is vital to keep TD under control promptly. To this end, several technical debt management (TDM) activities have been proposed to keep an acceptable level of TD accumulation on software systems (Li et al., 2015). Such activities include identifying TD items (e.g., by using source code analysis (Zazworka et al., 2013) or machine learning (Li et al., 2022)), documenting them (e.g., by opening issues in an issue tracking system (Kashiwa et al., 2022)), and paying back the debt (e.g., by refactoring the source code (Martini et al., 2015)).

Although keeping TD under control is paramount, TDM is time-consuming and can demand around a third of the development time (Besker et al., 2017, 2019). The more time is spent in dealing with TD, the less time is available for developers to deliver new code and value (through features) to costumers (Lim et al., 2012). This may partly explain why software development teams often do not perform any TDM activity despite being aware of the risks of TD (Rios et al., 2018; Martini et al., 2018). In such a context, where TDM is crucial to keep the overall quality of software systems, but developers cannot afford to spend significant time dealing with TD, the usage of tools to *efficiently* support TDM activities is of paramount importance (Khomyakov et al., 2019; Silva et al., 2022; Biazotto et al., 2023).

To explore how tools could support TDM more efficiently, researchers started to investigate practitioners' needs and elicit requirements for TDM tools. Examples of such requirements include: (i) automated tracking of TD (Martini et al., 2018); (ii) identification and monitoring of TD based on historical data (Martini et al., 2018); and (iii) linking TD-related information to refactoring opportunities (Sas et al., 2021; Malakuti and Ostroumov, 2020). However, this list is not exhaustive, and many other requirements remain implicit. Other works related to TDM in industry also focus on practitioners' understanding of TD (Codabux et al., 2017; Silva et al., 2019; Apa et al., 2020). Some studies also focus on investigating tools for TDM (Avgeriou et al., 2021), while others investigate the effort related to TDM (Besker et al., 2019). Finally, there is evidence that relates the level of TDM awareness with the usage of TDM tools.

Despite the advances on understanding practitioners' perspective on TD tools and TDM automation, the adoption of tools is still relatively low (Besker et al., 2019), which might compromise the efficiency of TDM. Besides, to the best of our knowledge, research has not yet investigated practitioners' reasons for the selection and usage of TDM tools (e.g., whether they prefer manual or automated execution), and it has not produced an extensive list of tool requirements that suits practitioners concerns, what might be one of the reasons for the lower levels of tool adoption.

To tackle the aforementioned problem, this paper aims at explaining how practitioners reason about adopting TDM tools and eliciting a comprehensive set of core requirements for such tools. This understanding can support the development of better TDM strategies and consequently make TDM more efficient. To this end, we conducted a survey with practitioners and received 103 responses. From those, 89 valid responses were analyzed and, based on them, we offer two main contributions:

- **Practitioners' rationale model for TDM tool adoption**: The model contains a set of nine concepts and their relations and helps explain what practitioners consider when selecting and adopting tools for supporting TDM. This model can support: (a) the development/improvement of tools that align with practitioners' concerns; and (b) the development of new strategies/techniques to support TDM, potentially increasing the automation level of TDM;
- **Set of core requirements for TDM tools**: This set contains 46 requirements and can be used to develop new tools or improve existing ones, potentially aligned with practitioners' needs/concerns;

The rest of this study is organized as follows. Section 2 describes background and related work and compares related work to our study, also elaborating on our main contributions. Section 3 reports the study design, covering the research questions and data collection and analysis. Section 4 presents the survey results, while Section 5 discusses them. Section 6 discusses the threats to the validity of this study and the actions we took to mitigate these threats. Finally, Section 7 concludes this work and highlights future research directions.

2 Background and Related Work

The way practitioners manage TD in an industrial context has drawn the attention of the TD research community since the early 2010s. Some of the essential results pertain to best practices to deal with TD (Rios et al., 2020), the leading causes and consequences of TD (Rios et al., 2018), and the consequences of TD on practitioners' productivity (Besker et al., 2018). Concerning best practices for TDM, Holvitie et al. (2014) conducted a survey with 54 practitioners to identify: (i) the practitioners' perceptions about TD; (ii) the agile development practices that help reduce TD; and (iii) the characteristics of TD instances. The results suggest that code standards are the most helpful agile practice; the authors also describe the main types of TD, the severity of TD, and the reasons for incurring TD. While this work focuses on investigating (agile) development processes, our study focuses on understanding the usage of tools in the development process and eliciting requirements to better support TDM.

Ernst et al. (2015) report on a survey with 536 respondents to explore their understanding of TD, the architectural decisions that are TD sources, and tools and practices that practitioners use to manage TD. The results report that half of the respondents do not use any tools, and the other half use static analyzers like SonarQube, Checkstyle, and Findbugs. Unlike Ernst et al.'s study, our study seeks to provide more contextual information, including the practitioners' reasons to adopt TDM tools and the potential requirements to improve such tools.

Codabux et al. (2017) report on a survey with 67 practitioners to explore their understanding of TD and the methods and metrics to measure TD, covering communication of TD and risk analysis. They also report that the most common method to measure TD is person/hour. The study also mentions technology to discuss TD items (e.g., SonarQube). Complementing this work, we aim to understand how technology is selected and the main concerns related to its usage.

Besker et al. (2017) surveyed 258 practitioners from 32 companies to answer research questions related to the time they spent dealing with TD, the negative impact of TD, and the activities that demand extra effort. In a similar direction, Besker et al. (2018) conducted a longitudinal study to understand the practitioners' perspective on the time spent due to TD. Their findings show that around one-third of all development time is spent on TDM, and all TD types, especially those related to complex architectural design, generate harmful consequences in software maintainability. Moreover, most of the extra time spent in TDM is dedicated to understanding and measuring TD items. Our study adds to the state of the art by further characterizing the concerns of practitioners while using tools to reduce the effort of TDM. Furthermore, we investigate requirements such as information provided by tools that can add value for supporting decision-making.

In another study, Martini et al. (2018) analyze how companies deal with TD and the level of correlation between awareness and the use of TDM tools. They also propose a model with five levels of tracking TD. Our work complements this work by providing more information on tools' requirements (e.g., related to information and interfaces), which could help the interactions between practitioners and tools and potentially enable higher levels of automation and efficiency.

Silva et al. (2019) and Apa et al. (2020) report similar studies considering answers collected in Brazil and Chile, respectively, focusing on which TDM activities practitioners perform and the technologies and tools they use. Compared to these studies, we provide more in-depth evidence regarding the requirements and rationale for the tools' adoption.

Avgeriou et al. (2021) examined tools for TD measurement considering three TD types (code, design, and architectural TD). The authors explore tools' features, popularity, and validation (i.e., whether or not tools are used in the industry). While their study focuses on identifying and reporting tools, our work focuses on investigating the concerns related to the usage of tools in an industrial context, adding the practitioner's point of view about TDM tools. Furthermore, we also elicit a set of requirements for tools, which is a new contribution compared to this work.

Other recent literature (Silva et al., 2022; Biazotto et al., 2023; Jeronimo Junior and Travassos, 2022) states that stakeholders use isolated TDM tools. At the same time, Jeronimo Junior and Travassos (2022) point to the need for easier integration of those tools within the current development toolset, towards a more holistic management of TD (i.e., performing more TDM activities and supporting them with tools). In the same direction, Biazotto et al. (2023) support the necessity of integrating automation artifacts to increase the level of automation in TDM, which could reduce developers' workload during TDM. These recent contributions present theoretical aspects of tool usage but lack a practical perspective.

Our study adds to the body of knowledge on TDM automation by investigating the practitioners' perspective on tools for TDM, specifically focusing on developers' rationale for tools' adoption and the requirements for such tools. These results can lead to improvements on existing tools (e.g., by implementing new features) and also the development of new strategies for TDM automation aligned with practitioners concerns about tool usage.

We summarize the related work in a model (see Figure 1), which helps discuss the current understanding of how practitioners use tools to support TDM. It also supports the understanding of the research design, specifically to develop the survey instrument (see Section 3.3). To build this model, we summarized the main results of each related work, subsequently analyzed them, and defined concepts and relations using constant comparison Strauss and Corbin (1990). For example, Avgeriou et al. (2021), Silva et al. (2022), Biazotto et al. (2023) present the information the tools provide, which helped us to define the concept Information on the model. In the same direction, Ernst et al. (2015) argue that practitioners use Information provided by tools for decision-making; this helped us to define the relation between Information and Practitioner.

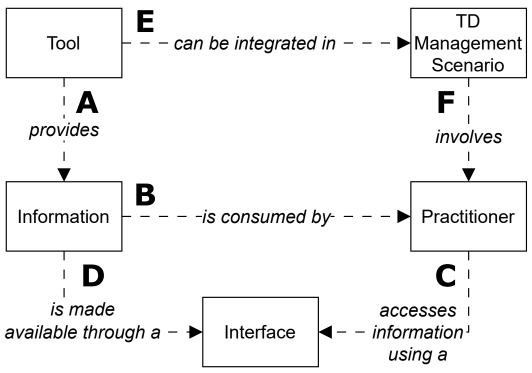


Fig. 1: Model for tool usage in software development

Tools produce and provide Information (e.g., TD accumulation) about software projects (**A**). Such information is consumed by Practitioners and supports their decision-making process (**B**). Practitioners access such information provided by tools using an Interface (**C**), which makes that information available (**D**). The interface refers to how information is presented to stakeholders (e.g., via a dashboard or e-mail). Tools could be integrated and executed in a TDM Scenario (**E**); such a scenario typically represents the communication between quality measurement tools (e.g., SonarCloud) and development tooling (e.g., Travis CI or IntelliJ IDEA). A scenario involves one or more practitioners (**F**) who interact with the tools using an Interface and make decisions based on the information provided.

3 Study Design

Considering the main goal of this study (i.e., explanation of how practitioners reason about adopting TDM tools and elicitation of a comprehensive set of requirements for such tools) and the guidelines for selecting empirical methods in software engineering research (Easterbrook et al., 2008), we selected the survey method as the most suitable to achieve our goal. Specifically, surveys are suitable for studies that "*ask about the nature of a particular target population*" (Easterbrook et al., 2008). In our case, the population is software practitioners, and we need to collect their opinions on how they reason about and what they require from TDM tools. We followed the guidelines for survey design proposed in (Kitchenham and Pfleeger, 2008), which includes:

- Step 1: Setting the objectives and research questions (detailed in Section 3.1)
- Step 2: Designing the survey (Section 3.2)
- Step 3: Developing the survey instrument (Section 3.3)
- Step 4: Obtaining valid data (Section 3.4)
- Step 5: Analyzing the data (Sections 3.5 and 3.6)

Figure 2 illustrates the overall research method, including its steps, inputs, and outputs. Since we have different data analysis for our two research questions (RQ_1 and RQ_2 presented in Section 3.1), Step 5 is decomposed into two steps (i.e., Steps 5.1 and 5.2). Finally, we describe the pilot study conducted to validate the data collection instrument and the data analysis methods (Section 3.3).

3.1 Step 1: Setting the objectives

Taking into account our research problem and the gaps presented in Section 2, the objective of this study, structured according to the Goal-Question-Metric template (van Solingen et al., 2002), is to *"**analyze** practitioners' opinions **for the purpose of** identifying requirements and rationale for adoption and usage **with respect to** TDM tools **from the point of view of** practitioners with varied roles **in the context of** both industrial and OSS development."* To achieve this objective, we defined the following research questions (RQ):

RQ_1 - How do practitioners reason about the adoption and usage of tools for TDM? In this RQ, we investigate the decision-making process behind the adoption of tools for TDM (i.e., how practitioners select tools), including factors regarding the tools' usage (e.g., how practitioners prefer to execute certain tools). The practitioners' reasons can provide insights into the perceived benefits of such tools, their concerns, and challenges to an effective tools adoption.

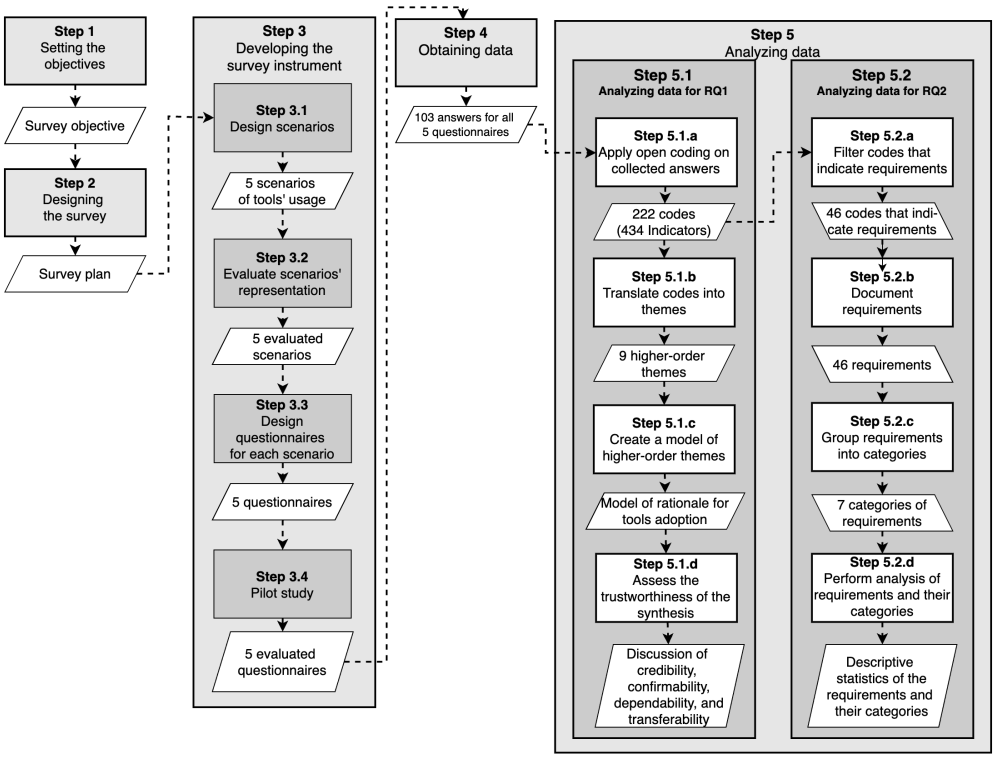

Step 1
Setting the objectives
Survey objective
Step 2
Designing the survey
Survey plan
Step 3
Developing the survey instrument
Step 3.1
Design scenarios
5 scenarios of tools' usage
Step 3.2
Evaluate scenarios' representation
5 evaluated scenarios
Step 3.3
Design questionnaires for each scenario
5 questionnaires
Step 3.4
Pilot study
5 evaluated questionnaires
Step 4
Obtaining data
103 answers for all 5 questionnaires
Step 5
Analyzing data
Step 5.1
Analyzing data for RQ1
Step 5.1.a
Apply open coding on collected answers
222 codes (434 Indicators)
Step 5.1.b
Translate codes into themes
9 higher-order themes
Step 5.1.c
Create a model of higher-order themes
Model of rationale for tools adoption
Step 5.1.d
Assess the trustworthiness of the synthesis
Discussion of credibility, confirmability, dependability, and transferability
Step 5.2
Analyzing data for RQ2
Step 5.2.a
Filter codes that indicate requirements
46 codes that indicate requirements
Step 5.2.b
Document requirements
46 requirements
Step 5.2.c
Group requirements into categories
7 categories of requirements
Step 5.2.d
Perform analysis of requirements and their categories
Descriptive statistics of the requirements and their categories

Fig. 2: Research Method

RQ$_2$ - What are the requirements for tools that can support TDM? Practitioners can use information provided by tools for TD-related decision-making (e.g., prioritizing TD items to be repaid). Moreover, practitioners can use a variety of interfaces (e.g., dashboards and e-mail) to query such information. To complement the results of RQ$_1$ (which focuses on the rationale for selecting and adopting tools), this RQ focuses on features (i.e., requirements) expected in TDM tools by practitioners. The answer to this RQ can support improving existing tools or developing new ones that fulfill the requirements.

3.2 Step 2: Designing the survey

According to Kitchenham and Pfleeger (2008)'s guidelines, there are two common types of surveys: (i) cross-sectional, which is a study that gathers information from participants at a single time; and (ii) longitudinal, which tracks changes within a population over time. Our study conducted a cross-sectional survey because practitioners base their opinions about tool usage on past experiences, a context in which such type of survey is suitable (Kitchenham and Pfleeger, 2008). Moreover, a cross-sectional survey is efficient in terms of time and resources since it enables the collection of opinions of practitioners (with various profiles from industry and OSS development) in a single iteration.

3.3 Step 3: Developing the survey instrument

This section describes the Steps 3.1 to 3.4 (shown in Figure 2) to design the questionnaires to collect data.

As presented in previous studies (Codabux et al., 2017), practitioners are aware of the TD concept, but their understanding might differ from each other. To prevent misunderstanding, which could reduce the precision of practitioners' insights, we decided to design the questionnaires based on concrete scenarios (i.e., with examples of TD) instead of the concepts of the TD metaphor (e.g., principal, interest, interest rate). We expected that this strategy would improve the communication with developers and potentially lead to more and better insights from them.

To design the scenarios, we initially had to understand which TDM activities were more relevant to require potential tool support. According to recent literature, practitioners tend to perform six TDM activities (identification, communication, prevention, prioritization, repayment, and documentation (Apa et al., 2020)). Hence, these six activities were used as basis for our scenarios.

To design the scenarios covering those activities, we relied on evidence found in the literature (**Step 3.1** in Figure 2), such as the ones presented in Section 2. For example, Martini et al. (2018) show some requirements collected from practitioners, who tend to use issue tracking systems to document TD and point to the necessity of tools to support and guide identification and tracking of TD items recorded in issues. This need helped us to propose scenario S1: a bot that labels issues. All scenarios are aligned with the model in Figure 1, containing elements related to information and interfaces provided by tools. In S1, the *label* represents the information, while the *issue tracking system* is the interface.

To select which tools would be presented in each scenario, we revisited the studies of Biazotto et al. (2023) and Silva et al. (2022), which present tools for TDM. We analyzed the tools considering for instance how they could be integrated with other technologies commonly used in modern software development (e.g., CI managers and code review platforms). For instance, SonarQube[1] was reported as one of the main tools for TDM (Silva et al., 2022) and can connect to Grafana[2], which is a well-known open-source dashboard. Based on such evidence, we proposed S2, which shows an alternative to visualize TD information (provided by SonarQube) within Grafana.

The scenarios were initially represented using the dynamic diagram of C4 Model[3]. We selected this type of diagram because it can capture both tools deployed as standalone (e.g., SonarCloud) and components deployed within other tools (e.g., a widget within Grafana Dashboard). It can also represent runtime interactions (e.g., an API call), which could help practitioners understand the proposed data/execution flow. To check if the representations were clear, we requested feedback from researchers in our group (**Step 3.2** in Figure 2). Their feedback pointed to difficulties in analyzing the diagrams due to the abstraction level. To tackle this problem, we decided to represent the scenarios using mockups, which we assumed could reduce the abstract nature of the diagrams. This is because presenting a real screen brings the solution

[1] https://www.sonarsource.com/products/sonarqube
[2] https://grafana.com/
[3] https://c4model.com/

closer to practitioners' usual interaction with tools (i.e., such interaction does not rely on architectural models as presented in our initial representation). The mockups were evaluated in pilot study (see Section 3.3), from which we noted a better understanding of the scenarios. The final version of S2's representation is presented in Figure 3 to exemplify the mockups. Due to space constraints, details about all five scenarios (including representations using C4) are omitted here and can be found in our replication package[4].

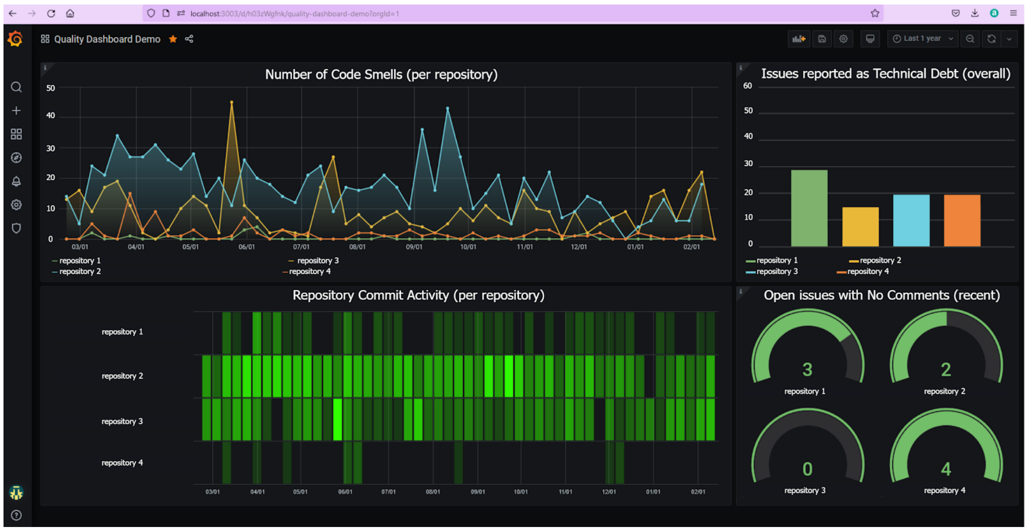

Quality Dashboard Demo
Number of Code Smells (per repository)
Issues reported as Technical Debt (overall)
Repository Commit Activity (per repository)
Open issues with No Comments (recent)
repository 1
repository 2
repository 3
repository 4
Last 1 year
3
2
0
4

Fig. 3: Example of S2's mockup

It is worth highlighting that we cannot claim that those scenarios capture all possible usages of tools. Our goal is to provide examples closer to modern software development activities in practice and can drive improvements on existing tools and strategies for TDM. We also acknowledge that using such scenarios might pose threats to the validity of this study that were mitigated using several strategies (see Section 6).

We designed five scenarios to support the data collection for this study (in parentheses, we state which TDM activities each scenario represents):

- *S1 (identification, documentation, and prioritization):* As stated by Martini et al. (2018), one of the requirements posed by practitioners is the automated tracking of TD items. The authors also state Jira is one of the most cited tools to support this process. Combined with this information, several machine learning models have been used to identify TD items using natural language (Li et al., 2022). Consequently, an automated solution for identifying TD in Jira issues and labelling them could be useful for prioritizing TD items to be repaid. Thus, S1 presents a bot that automatically labels issues within Jira.

[4]The replication package can be accessed on https://zenodo.org/records/13175210

- *S2 (communication):* Communicating information about TD is paramount for decision-making. The use of dashboards in software development can help the visualization of quality issues. Since TD can be related to quality issues, we assume that presenting information about TD in a dashboard would help keep TD items visible, especially in a visualization tool already used for maintenance purposes. S2 shows a dashboard that contains information that might help in TDM (see Figure 3).
- *S3 (identification and repayment):* Refactoring is a well-known task for paying back TD and is commonly performed by practitioners (Rios et al., 2020). There is evidence in literature stating that the suggestion of refactoring opportunities can help practitioners decide which TD items should be repaid (Martini et al., 2018). To support this process, plugins could be installed in IDEs to support the identification of code smells and suggest refactored code. Hence, S3 shows an IDE plugin that suggests refactoring actions.
- *S4 (communication):* Continuous integration and deployment (CI/CD) is a practice widely used in software development (Fitzgerald and Stol, 2017). This practice focuses on automating parts of the deployment process such as integration tests. Since this CI/CD is sometimes completely automated and happens every time new code is submitted to the production environment, integrating a TDM strategy as a step in the CI/CD process could be a good alternative to raise awareness about TD. In S4, we present a bot that submits reports to mailing lists after CI cycles.
- *S5 (identification and prevention):* Code review is a task performed in a new patch to ensure that it is correct to be merged into a software project. This task is performed in industrial and open-source projects, and some TD items can be identified during code review, preventing them from being inserted in source code (Kashiwa et al., 2022). S5 presents a bot that supports code review by providing a report about the amount of code smells in the new patch.

In **Step 3.3** (see Figure 2), we developed five questionnaires (one per scenario) using Google Forms[5] and distributed them online, aiming at reaching as many respondents as possible from our target population, i.e., software practitioners from both industry and open-source projects. Since the scenarios cover aspects commonly present in open-source settings (e.g., code review in pull requests) and industrial settings (e.g., dashboards), collecting opinions of practitioners who work in both settings helps enhance the generalizability of our study. The questionnaire starts with questions on practitioners' requirements and rationale regarding TD tools, followed up by open-ended questions to capture more insights and nuances on opinions about both requirements and rationale. An example of an open-ended question in S4 is: "*When would this type of notifications [about TD] be useful? For example, after each CI cycle, or if a criterion is met (e.g., the amount of code smells exceeds a threshold). Please justify your answer.*" With this question, we explore the requirements (condition for using the notification) and also the rationale (asking practitioners to justify the answer). Hence, we can collect data to answer RQ$_1$ (rationale) and RQ$_2$ (requirements).

[5] https://www.google.com/forms/

While the collected data is mainly qualitative, quantitative data was collected to evaluate the overall usefulness of each scenario (using Likert scale). According to our pilot studies, each questionnaire would take around 10 minutes to be answered. The full questionnaires can be accessed in our replication package[6].

In **Step 3.4**, we conducted a pilot study to assess our scenarios and questionnaires. Five researchers from our research groups (at the University of Groningen and University of São Paulo) participated, each assigned to evaluate one scenario. They were instructed to monitor the time taken to answer the questions and note any encountered issues, such as unclear questions or inadequate scenario representation. We note that the data collected during the pilot study were not included in the analyzed dataset (89 valid responses).

Based on the pilot findings, we noted that each scenario could be completed within approximately 10 minutes. This duration was considered reasonable to encourage more responses (because it is a short survey) and ensure that respondents remained engaged throughout the survey, yielding more meaningful insights.

Following the pilot, we implemented some changes to the scenarios representation and the questionnaires. First, we revised the phrasing of specific questions, removing the term "technical debt" due to feedback suggesting potential misunderstanding. Additionally, we introduced a new question in the questionnaire for S1: "*If not, are there specific circumstances (e.g., upon reaching a certain threshold) where notifications about issues would be beneficial?*" This addition was derived from feedback indicating that notifications could be valuable in other contexts not captured in that scenario, and no question related to S1 could capture such insight.

3.4 Step 4: Obtaining data

To recruit respondents, considering our population, the study used convenience sampling (Linåker et al., 2015), a non-probabilistic sampling in which it is not possible to observe randomness on the selected units from the population. Specifically, an invitation to participate and to further disseminate the survey was distributed by e-mail across the personal networks of the authors, to contributors of well-known OSS projects (e.g., from the Apache Software Foundation), and using LinkedIn private messages. We also posted the invitation on LinkedIn and X/Twitter feeds as an alternative to reach more practitioners.

3.5 Step 5.1: Analyzing data for RQ_1

For answering RQ_1, we used thematic synthesis (Cruzes and Dyba, 2011), a method employed for recognizing, interpreting, and presenting patterns (themes) within data in qualitative research. We performed five tasks, aligned with the guidelines summarized by Cruzes and Dyba (2011):

- **Extract data:** In our context, the data extraction step is the collection of responses, i.e., we extracted the data by collecting 103 responses, from which 89 were valid. Hence, there is an overlap between Step 4 of the research process

[6]The replication package can be accessed on https://zenodo.org/records/13175210

(Obtaining Valid Data) and this first sub-step of thematic synthesis. Thus, both activities are represented in Figure 2 as Step 4 (Obtaining Valid Data).

- **Code data:** We started applying open coding to break down practitioners' answers into smaller pieces of information, which are called "*indicators*" (**Step 5.1a** in Figure 2) Those are represented by a code (i.e., a word, phrase, label, etc.), which is used to navigate over different indicators and compare them using constant comparison. For instance, one answer from the survey states that: "*s1_r2: In general, such notifications should not become yet another source of useless spam everyone automatically ignores.*" During open coding, we assigned the code *1-notifications are not useful* to that indicator (the number 1 refers to the scenario), which helped us compare that indicator with other ones.

  We used an integrated approach (Cruzes and Dyba, 2011) to code the data. Initially, three authors coded a sample of 30 answers (which were randomly selected), generating as many codes as possible from the data. After that, they discussed the generated codes until reaching an agreement. Next, the first author and an external collaborator coded all the answers, using both the previously generated set of codes and proposing new codes, which were necessary to better represent indicators. After all data was coded by the first author and the external collaborator, all the four coders (the three authors that were involved in the open coding and the external collaborator) discussed the codes and reached an agreement on the indicators and codes to represent them. Considering that we have five different scenarios, we assigned the scenario number to the codes (e.g.,"**2-**dedicated td dashboard is useful" is a code in the second scenario) to help us to check if any of the scenarios is biasing the data analysis (i.e., a certain concept in the model is only supported by indicators form a specific scenario). The open coding sub-step resulted in a set of 222 codes and 425 indicators. Each scenario resulted in a different number of codes, i.e., *S1=33*, *S2=75*, *S3=28*, *S4=35*, *S5*. Table 1 presents a few examples of codes and indicators generated during the open coding sub-step.
- **Translate codes into themes:** A theme emerges through the process of coding, categorization, and reflective analysis. They represent abstract constructs that imbue recurring experiences with meaning and coherence. In this study, we adopted a grounded-theory-based approach due to its structured steps (open, selective, and theoretical coding) for translating the code into themes (**Step 5.1b** in Figure 2). The emphasis of this approach is constant comparison, which aids in mitigating potential biases and enhancing data exploration. It is important to note that while we followed a grounded theory-based approach, we did not adhere strictly to all its steps and, therefore, the resulting themes cannot be claimed as a fully developed theory. Interactive data collection and analysis were not employed, thus theoretical saturation cannot be guaranteed, a point further discussed in Section 6.
- **Create a model of higher-order themes:** The themes that emerged in the previous steps were further explored and interpreted to create a model consisting of higher-order themes and the relationships between them (**Step 5.1c** in Figure 2). In this study, we propose a model that captures the main concerns that practitioners have while reasoning about tools' adoption (see Section 4.2).

- **Assess the trustworthiness of the synthesis:** Assessing the reliability of interpretations derived from thematic synthesis involves constructing arguments supporting the most plausible interpretations (**Step 5.1d** in Figure 2). Research findings do not have a singular correct interpretation but rather the most probable interpretation from a specific standpoint. Therefore, enhancing trustworthiness entails presenting findings in a manner that encourages readers to consider alternative interpretations. To evaluate the reliability of the results, we examined the concepts of credibility, confirmability, dependability, and transferability. The assessment of trustworthiness will be detailed in Section 6.

Table 1: Example of coding process

answer	**author 1**	**author 2**	**author 3**	**init. agree.**	**extern.**	**final code**
Within IDE for my own code, an external tool for code reviews.	External tool for refactoring suggestions for code review — Within IDE for refactoring suggestions for own code	internal-own-code — external-support-code-review	A plugin within the IDE helps during code development — An external tool helps during code review	External tool helps during code review — Internal tool helps with own code	s3_within the IDE preferred for own code	3-ide plugin is useful for own code — 3-external tool is useful for code review
In my experience, most of the code smells detected by SonarQube were 'false' code smells, like multiples if's or functions with more then x amount of parameters, those will never be 'fixed', so this maybe not reflect the actual quality of the code.	SonarQube code smells are mostly false positives	disbelief-quality-accuracy	Detections tools results quality is a concern — Not all detected smells should be fixed — Due to low accuracy of detection, the reports may not reflect a system's quality	disbelief-quality-accuracy	s4_false code smells in SonarQube	4-accuracy of identified code smells is a concern

init. agree.: Codes agreed after the coding by the first three authors.
extern. collab.: Codes generated by the external collaborator.

3.6 Step 5.2: Analyzing data for RQ_2

To answer RQ_2, we elicited a set of requirements for tools that can support TDM. For eliciting such requirements, we reviewed the codes/indicators previously create during open coding for RQ1 (Step 5.1a in Figure 2) and filtered those presenting information that could be translated to a requirement (**Step 5.2a** in Figure 2). For instance the code "*2-group smells by type*" represents indicators such as "*s2_r8: maybe* *a classification of which types of code smells are more common in each repository*" and "*s2_r13: A list with smells grouped by type*". Such indicators point to a feature that could be implemented in a dashboard, thus it was included for the next step of the requirements elicitation process. For this step, we considered all codes generated during open coding, even those with few indicators, because our goal was to report any requirement pointed out by practitioners. Although requirements pointed out by few or even a single respondent have a low level of consensus within our sample, such requirements may represent concerns common to other practitioners who were not surveyed.

After filtering codes and indicators, we documented the requirements using Pohl (2016)'s template (**Step 5.2b** in Figure 2), as presented in Figure 4.

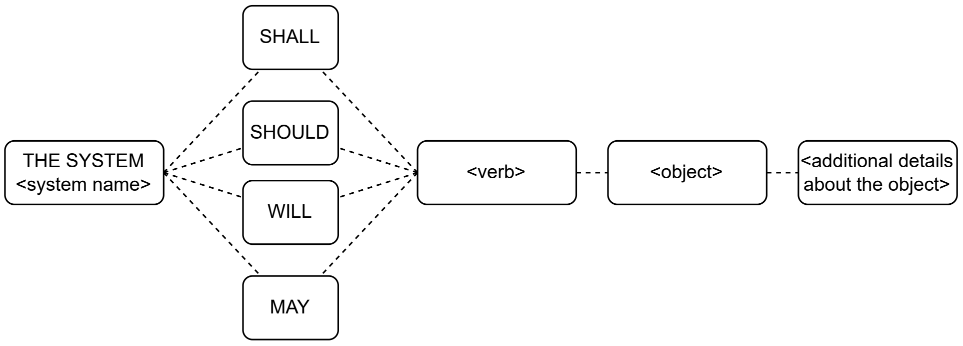


Fig. 4: Template for documenting requirements using natural language (Adapted from Pohl (2016))

The template is composed of five elements:

- **system name:** It indicates the tool presented in the scenario from which the requirement was extracted;
- **modal verb:** A modal verb that indicates the the obligation for each requirements, i.e., legally obligatory requirements (*shall*), urgently recommended (*should*), future requirements (*will*), and desirable requirements (*may*). We decided to use the verb "*will*" to document all the requirements. This is because we cannot prioritize the requirements from the data we have available. We can, however, claim that all requirements are future requirements. It is worth highlighting that these requirements must be prioritized in the future, and we discuss this in Section 5;
- **verb:** It refers to the core of each requirement, i.e., the functionality that it specifies (e.g., present, enable, notify, etc.). The verb depicts the system behavior by means of a requirement;
- **object:** Some verbs need an object to be complete. For instance, the verb "*present*" is amended by the information of *what* is being presented and *where* it is presented; and

- **additional details about the object:** It refers to details about the object that makes the requirement clearer, for instance, the font-size of the printed text.

Considering the previous example ("*2-group smells by type*"), we identified the following elements: (a) a dashboard is the **system**, since it was the example presented in S2, from where the code/indicators were collected; (b) the **modal verb** is "*will*", since it is a future requirement; (c) the **verb** is "*present*", since the list of smells is presented in a dashboard; (d) "*a list of code smells grouped by type*" is the **object** and represents "*what*" may be presented. Finally, we did not identify additional information for this requirement. Grouping all the elements of the model, we ended up with the following requirement: "*The dashboard will present a list of code smells grouped by type.*"

We chose this template for three main reasons. First, it is specific for documenting requirements using natural language, which is the data we have available to elicit requirements. Second, it is a well-known template, widely accepted and cited by the requirements engineering community (it was cited more than 600 times on Google Scholar). Finally, it enables us to map several aspects of the requirements (e.g., system and process), enhancing the replicability of the process.

To enable a better analysis and discussion of the requirements, we also grouped them into categories and sub-categories, considering the verb and object in each requirement (**Step 5.2c** in Figure 2). We then used descriptive statistics to analyze both the requirements (e.g., number of respondents supporting each requirement) and the categories (e.g., number of requirements per category), which is represented in **Step 5.2d** in Figure 2.

4 Results

4.1 Demographics of respondents

Considering all scenarios (i.e., S1–S5), we received 103 responses. From such answers, 11 were blank and were excluded. Furthermore, considering the population of this survey, i.e., software practitioners, we excluded three more answers, in which people informed their roles as PhD Student, Student, and Researcher. Thus, we considered 89 valid responses to address our RQs.

Each scenario is independent, thus practitioners could fill in questionnaires for one or more scenarios, which resulted in different amounts of responses per scenario, i.e., S1 had 21 responses (n=21), S2 n=18, S3 n=18, S4 n=11, and S5 n=21. Table 2 shows the demographics per scenario and practitioners profile characteristic (i.e., role, experience, and education). It is worth highlighting that the responses were anonymous, and we cannot determine how many individuals participated in the survey. Hence, the demographics can be interpreted as the amount of responses provided by practitioners with a certain profile characteristic.

The practitioners were asked about their roles in software development. To collect their roles, we provided a set of options based on previous surveys related to TD in industry and OSS (Rios et al., 2020) (i.e., Software Architect/Designer, Software Developer, Project Manager, Quality Assurance). Practitioners could also indicate their roles in an "*Other*" field if the pre-defined roles did not describe their current position. According to Table 2, around 66% (59/89) of the responses were provided by software developers, while software architects/designers accounted for 21% (19/89). Other roles, such as costumer engineer and project managers, also responded to our survey.

Table 2: Number of respondents per profile characteristic per scenario.

	Characteristic	**S1 n=21**	**S2 n=18**	**S3 n=18**	**S4 n=11**	**S5 n=21**	**Total n=89**
Role	Software Developer	13	12	11	7	16	59
	Software Architect/Designer	6	4	4	2	3	19
	Project/Engineering Manager	2	2	2	2	2	10
	Costumer Engineer	0	0	1	0	0	1
Experience	more than 10 years	13	9	11	7	13	53
	6-10 year	1	3	3	1	3	11
	1-5 year	7	6	4	3	4	24
	less than 1 year	0	0	0	0	1	1
Education	Unfinished Bachelor	2	2	2	2	2	10
	Bachelor	7	6	7	5	9	34
	Master	8	7	6	2	5	28
	PhD	4	3	2	2	4	15
	Other	0	0	1	0	1	2

Regarding practitioner's experience, more than 72% of the responses (64/89) were provided by practitioners with more than 5 years of experience, with around 60% of the responses (53/89) were provided by those with more than 10 years of experience. Since many responses come from experienced practitioners, we expect the derived insights to also reflect this characteristic. Nonetheless, since newcomers, with less than one year of experience, are under-represented (around 1% of the responses), requirements that support the needs of these individuals may be lacking.

Most responses were provided by graduates of bachelors (34/89) or Masters (28/89) degrees. Moreover, 11% of the responses (10/89) were provided by practitioners who have not finished their bachelor's degree. Since most respondents have an academic background, we conjecture that they have some understanding of the TD metaphor (e.g., what is TD, what causes TD, how to measure TD, etc.). This is because the TD metaphor is usually covered in Software Engineering courses (Avgeriou et al., 2023).

We also investigated the overall perception about the scenarios' usefulness. The results were collected with a Likert scale related to the following statement: "*I consider this scenario useful*". Overall, most practitioners consider S1, S2, S3, and S5 useful, as presented in Table 3. For S4, the same amount of practitioners consider it useful and not useful. Among all scenarios, S3 is the one considered most useful, with 9/18 practitioners strongly agreeing with the statement. This might indicate that the support for paying back TD (which is represented in S3) is a key concern for practitioners. S4 and S5, on the other side, are the ones with the highest number of practitioners disagreeing or strongly disagreeing with the statement (total of 5 = 3+2). In summary, our first finding is the following:

Table 3: Perceived usefulness of the scenarios, presented as a Likert scale.

scenario	I consider this scenario useful				
	Strongly agree	**Agree**	**Neutral**	**Disagree**	**Strongly disagree**
S1	5	8	4	3	1
S2	7	5	3	1	2
S3	9	3	4	2	0
S4	2	2	2	3	2
S5	7	8	0	3	2

Finding #1: Overall, the responses were provided by practitioners with a variety of roles, years of experience, and educational backgrounds. *Software developer* is the most common role, while more than 68% of the responses were provided by experienced practitioners. Moreover, S3 was considered more useful by practitioners, indicating that the support for paying back TD is a key concern; thus, tools' features that support such activity could be prioritized.

4.2 RQ$_1$ - Tool adoption rationale

Overall, we aimed at understanding how practitioners reason about tools for automating TDM and identifying factors that influence decision-making processes regarding tool selection and usage. Figure 5 presents a model derived from the answers and composed of 10 themes and their relationships.

Both themes and relationships were defined using constant comparison. We compared codes and indicators to define themes, grouping them based on their main topic. For instance, consider the following indicators: "*s3_r8: A patch needs to be applied to get reviewed, because auto-patches (from my experience) always require human validation,*" and "*s1_r8: Auto-labeling in principle seems a bit interesting, but I tend to think that issue triage is best done as a human process.*" When analyzing and reflecting on these indicators, we observed that they both highlight the need for humans to review tools' execution and outputs, guiding us to define the theme of **Human Intervention**.

The relationships were defined during Step 3 of thematic synthesis (5.3a in Figure 2). Essentially, a relationship is defined and supported in two cases: (a) two or more indicators in the same answer point to two different themes suggesting a link between them, and (b) a single indicator is related to two different themes, also pointing to a link between them. To exemplify these two cases, consider the relationship between **Human Intervention** and **Tool Quality**:

(a) the indicators "*s1_r4: What label should the bot use in that case? Can it assess the situation properly and propose more than 1 labels?*" and "*s1_r4: users might want the bot to wait for confirmation on the labels*", come for the same answer and are related to **Tool Quality** and **Human Intervention**, respectively.

(b) the indicator "*s3_r7: Depends on what such refactoring proposal tool actually does. One of my colleagues seems to blindly accept all of those and the result is a pure DRY code that is hard to read.*" is related to both **Human Intervention** and **Tool Quality**, because it indicates that code proposed by tools are usually "dry" (i.e., low quality) and must be manually reviewed.

The examples in both cases support the relationship between **Human Intervention** and **Tool Quality**, indicating that practitioners intervene on tools execution and output due to concerns regarding the quality of the tool.

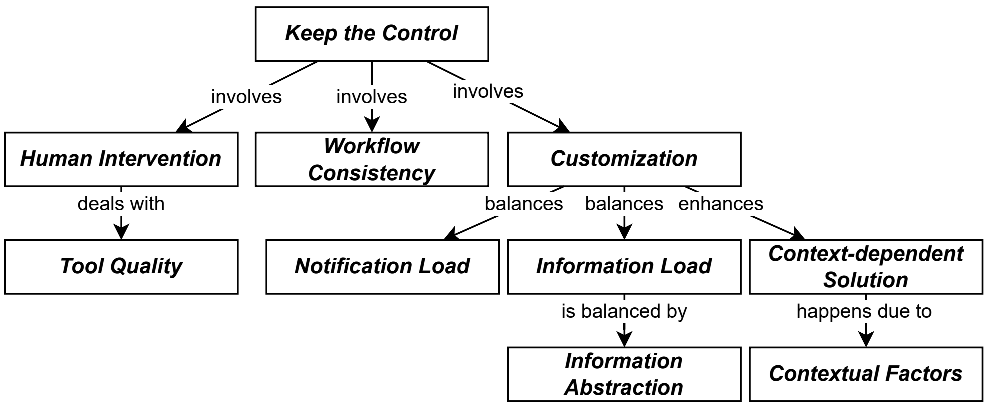


Fig. 5: Model of practitioners' rationale of tool adoption

The central theme on the model is **Keep the control**, a theme that conveys the desire of developers to maintain authority and oversight over the execution and outputs of tools used for TD management. It reflects a context in which developers acknowledge the benefits of automation but want to retain the control over when, how, and what changes are applied to their code by tools. This reflects, for instance, a preference for manual intervention in the process rather than complete automation: "*s3_r14: I prefer to execute at my own discretion, simply because I prefer to have more control over my code.*" Besides, the theme also points to a preference for human judgment over automated processes, especially when it comes to code changes: "*s3_r18: Leave the decision of accepting the suggestion to the developer and don't fill Git with automated patches.*" Overall, developers emphasize the importance of human judgment and intervention in the development process, even in the face of increasing automation.

To help practitioners keeping the control over tools, **Human Intervention** is highly used. It refers to the practice of manually executing tools rather than relying solely on automated solutions: "*s1_r8: Auto-labeling in principle seems a bit interesting, but I tend to think that issue triage is best done as a human process.*" The theme emphasizes the importance of human judgment in tasks such as issue triage, labeling, prioritization, and reviewing code changes. The following types of intervention could be identified:

- **Manual issue labeling and triage:** practitioners prefer to add labels to tickets related to TD manually, recognizing the importance of accurate labeling for effective organization and reporting: "*s1_r3: however if the ticket is related to TD, the creator itself can add the label. It is not so realistic that the description mentions tech debt, but the label is not added, especially if reports/filters are depending on it.*"

- **Issue prioritization:** Practitioners express a preference for human-driven prioritization over automated solutions, citing humans' superior ability to assess and prioritize issues effectively: "*s1_r8: Prioritization should be done by the team or the individual that owns a component.*"
- **Choosing information to be visualized:** Practitioners claim that it is important that a tool enables the selection of which information is presented: "*s2_r20: the dashboard could get bloated if the number of repositories get too big, but you could filter by a specific project.*"
- **Obtaining and reviewing refactoring suggestions:** Reviewing refactoring suggestions is also necessary to tackle potential mistakes that tools can make: "*s3_r3: I would prefer an IDE plugin and would not prefer automatic patches AT ALL. I may not want to apply the suggestions, and the tool can also make mistakes. I want the control over when, how and what I apply.*"

Some practitioners claim that **Tool Quality** is one of the main concerns that lead to the need of human intervention. Practitioners are often skeptical about the quality of suggestions provided by tools: "*s1_r11: Not useful for me, as I'd not trust the bot to properly label the issue by simply looking at the description.*" Besides, they also claim that suggestions provided by tools are often too simplistic, and it is not worth developers' time to review them: "*s3_r2: In my experience such [refactoring] suggestions are either too simplistic, miss the mark or don't detect most interesting situations worthy of developer attention.*" Overall, the practitioners claim that the usefulness of a tool depends on how good its output is: "*s5_r8: This [usefulness] would obviously depend heavily on the quality of the reported problems.*"

Another concern regarding the quality of tools is the performance, which must be good enough: "*s3_r5: Proactively [providing refactoring suggestions] since it'd be good to see suggestions on the fly. The performance must be good though.*". Performance is also a concern due to potential disturbance on developers workflow (e.g., practitioners have to stop coding to wait for automated suggestions). To avoid such disturbances, practitioners would prefer for the tool to be executed later on the development cycle (e.g., at commit time) and not in real-time (e.g., while practitioners are coding): "*s3_r18: Proactively if not at a relevant performance cost. Otherwise it could be triggered at commit time.*" This points to another concept: **Workflow Consistency**.

The **Workflow Consistency** refers to how easily the TDM tools are integrated with existing tooling (e.g., IDE), and the level of disturbance that the TDM tool causes on the current workflow: "*s2_r15: Moreover, even if you propose them [practitioners] a valid solution, they might not want to because it might alter their workflow.*" The consistency of workflow can be achieved with two strategies. In the first case, developers use TDM tools within other tools already used by them (e.g., IDEs): "*s3_r14: I don't like to install external tools when working on an IDE.*" This means that practitioners tend to centralize tooling (including TDM tools) to avoid having many tools to manage: "*s3_r16: At code level I will prefer within the IDE primary due to its usability features, such as Copilot, CodeGPT, and similar aids.*" The second strategy to keep workflow consistency is using steps on the CI cycle, instead of installing and managing TDM tools individually: "*s5_r12: Another option [instead of executing a bot] is to* *have a step in CI so it can inform or break the build (depending on the context / type of project).*"

Keeping the control over tools execution and outputs also involves **Customization**, a theme that indicates practitioners' concerns regarding tailoring tools for their development environment and workflow: "*s1_r2: The idea [of automated issue labeling] sounds good on the surface, but I very much doubt it would be useful without extensive intelligent customization options.*" The customization options involve many aspects (e.g., tools results, people to notify, etc.) and vary in different scenarios. For instance, one of the main concerns in S1 is about the frequency of notifications (also see **Notification/Information Load** below): "*s1_r22: You should be able to configure queries and notifications based on the labels, for example.*". Similarly, in S4, one concern related to customization is what threshold should trigger reports: "*s4_r10: I'm not too fond of e-mail for this kind of checks [code smells checks], but if I need to consider e-mails I'd appreciate a setup in which I get an e-mail after a threshold for a project I'm interested in is breached and a summary of the tracked metrics once every week.*" Overall, tailoring tools helps practitioners to extract more value from TDM tools: "*s1_r21: Yes, depending on the label it would be cool [to have such a bot that labels issues], maybe just add a control so it doesn't notify for not so important labels?*".

The need for configuration options to a large extent points to another concern: the amount of notifications and information provided by the tools, leading to the themes **Notification Load** and **Information Load**, respectively. Regarding **Notification Load**, it refers to the quantity of notifications sent by tools (e.g., sending e-mails that report problems during CI). Practitioners highlighted that very often, the amount of notifications is too high and can cause stress: "*s4_r7: Today I turn off any kind of automatic notifications to not become stressed.*" Besides, too many notifications can distract developers, leading them to ignore even the important ones: "*s4_r1: CI are notoriously hard to get all green reliably, another source of constant CI notifications would distract from what actually matters.*"

Information Load relates to the amount of information and suggestions provided by tools. Practitioners claim that dashboards presenting many different pieces of information are not very useful: "*s2_r16: Depends on the UI, but if the UI isn't too bloated, maybe widgets and graph would be fine.*" Overall, practitioners are interested on receiving less information, but more valuable pieces of information: "*s2_r19: Maybe you can create an algorithm to suggest a bite-sized list of smells that could be tackled next.*"

To deal with information load, practitioners tend to summarize information to facilitate the analysis, which refers to the theme of **Information Abstraction**. For example, rather than dealing with long lists of code smells, practitioners prefer to analyze code smells in groups and then identify areas of interest on the project: "*s2_r12: [group by package/file] because the package/file division would help on prioritize the issues on focusing the team efforts in specific parts of the software.*" Also, summarizing information in a temporal basis is a preference to check the evolution of the project's overall quality:"*s4_r6: More than time, it would be desirable to see the evolution of the numbers over time to show progress/degradation of the software.*"

One more way to support customization is through the theme of **Context-dependent Solution**, indicating that practitioners find it impossible to have a solution that fits all companies: "*s2_r15: Every company has its own metrics and its own ways of keeping track of issues and technical debt (some don't even do it). Therefore I think it's very hard to find a solution that fits all.*" A **Context-dependent Solution** is required to meet the varying demands from different organizations. For instance, practitioners claim that: "*s1_r4: That [a notification about a TD label] would make sense in the scenario where a developer or a company is highly focused on a particular feature. Adding a label that notifies the manager and the developer on an issue around that feature would definitely help them.*"

Finally, a context-dependent solution is highly impacted by **Contextual Factors**, which characteristics of the company, team, and project that determine how a tool should be tailored. For instance, the maturity of a team impacts the usefulness of TD labels: "*s1_r22: the bot offering the label might be good to help teams that don't tackle tech debt yet*". Another factor that impacts the tool usage is the severity of the issue: "*s1_r18: This is a matter I consider highly context-dependent. Notification would be very useful in critical scenarios that require immediate actions. Otherwise, I don't find it as useful because issues are typically monitored with some frequency in software development projects.*"

We analyzed how many indicators per scenario support each one of the themes, as presented in Table 4. According to the table, the most common theme is **Information Abstraction**, which is supported by 34 indicators. Also, **Information Load**, **Customization**, and **Tool Quality** are supported by indicators from all five scenarios, indicating that such concerns are always present, regardless of the tools used.

Table 4: Number of indicators that support each theme, organized by the scenario from which the indicators were extracted.

Theme	**Number of indicators**					
	S1	**S2**	**S3**	**S4**	**S5**	**Total**
Information Abstraction	5	26	0	1	2	34
Workflow Consitency	0	3	21	2	1	27
Customization	10	2	2	1	12	27
Human Intervention	6	3	14	0	2	25
Tool Quality	4	5	10	1	3	23
Notification Load	10	0	0	5	7	22
Contextual Factors	10	2	0	0	2	14
Information Load	2	6	2	1	2	13
Context-Dependent Solution	4	3	1	0	2	10
Keep the Control	2	1	4	0	0	7
Total	53	51	54	11	33	202

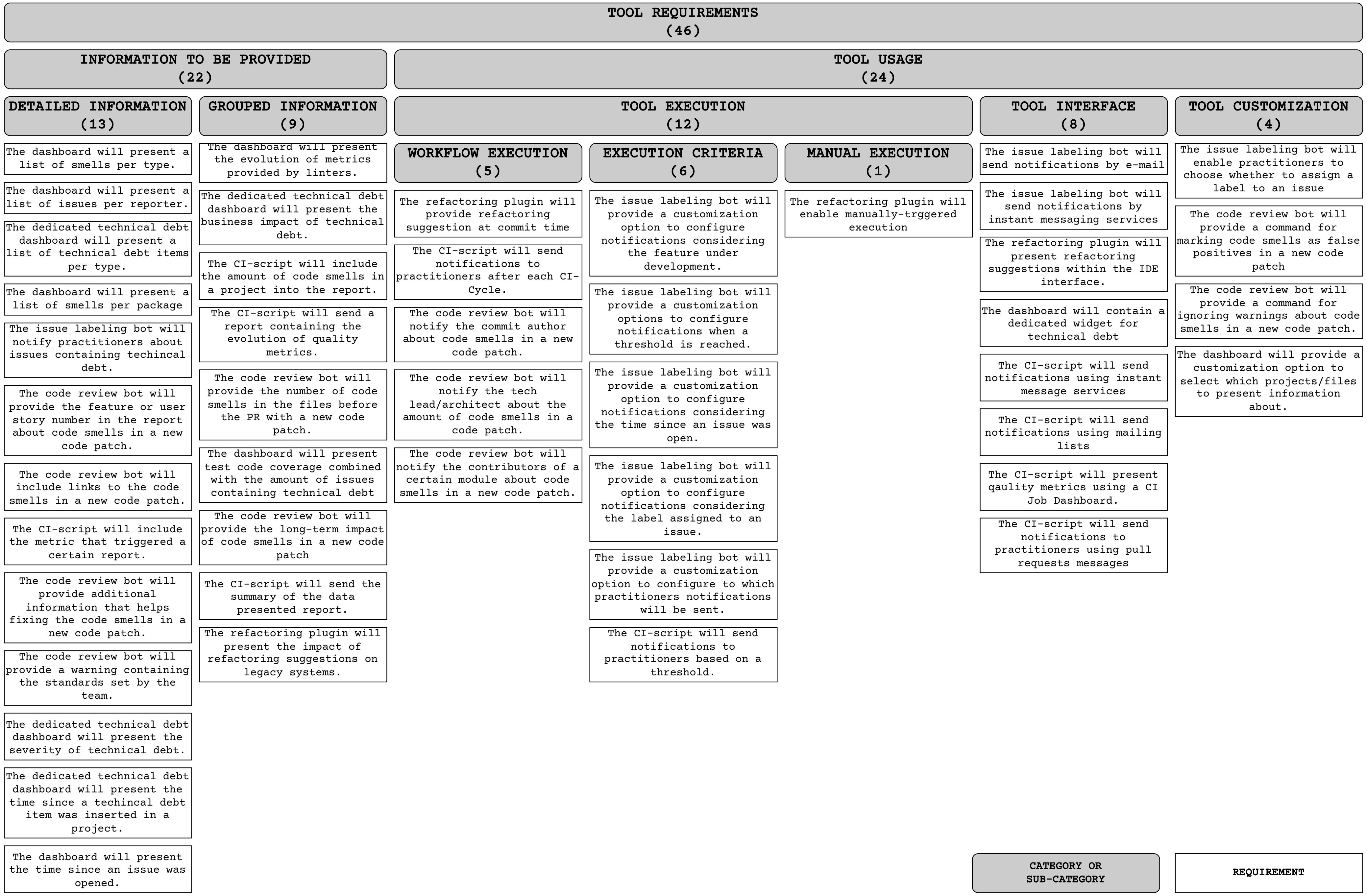
Analyzing how each scenario contributes to the model, we see that S1, S2, and S3 contribute with around 50 indicators each, while fewer indicators were collected from S5 (33 indicators). For S4, the lower amount of indicators (11 indicators) might be explained by the lower amount of responses (12 responses) compared to the other scenarios (which had around 20 responses each). Next, our second findings is:

Finding #2: While practitioners acknowledge that using tools benefits TDM, they have concerns regarding the tools' results and execution. Thus, when deciding to adopt the tools, being able to control different aspects of them is a crucial aspect. Furthermore, **Information Load**, **Customization**, and **Tool Quality** seems to be the most common concerns, since they were mentioned in all five scenarios.

4.3 RQ$_2$ - Tool requirements

To answer RQ$_2$, we elicited and documented 46 requirements for tools that support TDM. To facilitate the understanding of the requirements, we categorized them by comparing their verb and object (refer to Section 3.6 for more details). For example, in the requirements "*The dashboard will present a list of smells grouped by package*" and "*The CI script will include the amount of code smells in a project into the report,*" both objects refer to information that practitioners want to query from tools (i.e., list of smells and amount of code smells, respectively). Hence, we grouped them into the category "information to be provided". We further refined the categories into subcategories that could capture the nuances between the requirements in the same category. In the previous examples, "list of smells" belongs to *detailed information* about each smell in a project, and "amount of code smells" represents *grouped information*. Figure 6 presents the list of requirements and grouping structure. It is worth highlighting that the requirements and categories are related to the themes presented in RQ$_1$; we will discuss this relationship in Section 5. In the following, we present the requirements grouped according to the aforementioned categories.

Information to be provided/Detailed Information (13/46): This category includes tool requirements pointing to the need for information with finer levels of granularity. The category consists of requirements such as sending notifications for each issue with a certain label (e.g., "*The issue labeling bot will notify practitioners about issues containing technical debt*"), listing code smells (e.g., "*The dashboard will present a list of smells per package*"), and listing of issues (e.g., "*The dashboard will present a list of issues per reporter*"). Detailed information can enable practitioners, for instance, to compare code smells and prioritize them.



- TOOL REQUIREMENTS (46)
  - INFORMATION TO BE PROVIDED (22)
    - DETAILED INFORMATION (13)
      - The dashboard will present a list of smells per type.
      - The dashboard will present a list of issues per reporter.
      - The dedicated technical debt dashboard will present a list of technical debt items per type.
      - The dashboard will present a list of smells per package
      - The issue labeling bot will notify practitioners about issues containing techincal debt.
      - The code review bot will provide the feature or user story number in the report about code smells in a new code patch.
      - The code review bot will include links to the code smells in a new code patch.
      - The CI-script will include the metric that triggered a certain report.
      - The code review bot will provide additional information that helps fixing the code smells in a new code patch.
      - The code review bot will provide a warning containing the standards set by the team.
      - The dedicated technical debt dashboard will present the severity of technical debt.
      - The dedicated technical debt dashboard will present the time since a techincal debt item was inserted in a project.
      - The dashboard will present the time since an issue was opened.
    - GROUPED INFORMATION (9)
      - The dashboard will present the evolution of metrics provided by linters.
      - The dedicated technical debt dashboard will present the business impact of technical debt.
      - The CI-script will include the amount of code smells in a project into the report.
      - The CI-script will send a report containing the evolution of quality metrics.
      - The code review bot will provide the number of code smells in the files before the PR with a new code patch.
      - The dashboard will present test code coverage combined with the amount of issues containing technical debt
      - The code review bot will provide the long-term impact of code smells in a new code patch
      - The CI-script will send the summary of the data presented report.
      - The refactoring plugin will present the impact of refactoring suggestions on legacy systems.
  - TOOL USAGE (24)
    - TOOL EXECUTION (12)
      - WORKFLOW EXECUTION (5)
        - The refactoring plugin will provide refactoring suggestion at commit time
        - The CI-script will send notifications to practitioners after each CI-Cycle.
        - The code review bot will notify the commit author about code smells in a new code patch.
        - The code review bot will notify the tech lead/architect about the amount of code smells in a code patch.
        - The code review bot will notify the contributors of a certain module about code smells in a new code patch.
      - EXECUTION CRITERIA (6)
        - The issue labeling bot will provide a customization option to configure notifications considering the feature under development.
        - The issue labeling bot will provide a customization options to configure notifications when a threshold is reached.
        - The issue labeling bot will provide a customization option to configure notifications considering the time since an issue was open.
        - The issue labeling bot will provide a customization option to configure notifications considering the label assigned to an issue.
        - The issue labeling bot will provide a customization option to configure to which practitioners notifications will be sent.
        - The CI-script will send notifications to practitioners based on a threshold.
      - MANUAL EXECUTION (1)
        - The refactoring plugin will enable manually-trggered execution
    - TOOL INTERFACE (8)
      - The issue labeling bot will send notifications by e-mail
      - The issue labeling bot will send notifications by instant messaging services
      - The refactoring plugin will present refactoring suggestions within the IDE interface.
      - The dashboard will contain a dedicated widget for technical debt
      - The CI-script will send notifications using instant message services
      - The CI-script will send notifications using mailing lists
      - The CI-script will present qaulity metrics using a CI Job Dashboard.
      - The CI-script will send notifications to practitioners using pull requests messages
    - TOOL CUSTOMIZATION (4)
      - The issue labeling bot will enable practitioners to choose whether to assign a label to an issue
      - The code review bot will provide a command for marking code smells as false positives in a new code patch
      - The code review bot will provide a command for ignoring warnings about code smells in a new code patch.
      - The dashboard will provide a customization option to select which projects/files to present information about.

Fig. 6: List of elicited requirements and the categories and subcategories of requirements.

Information to be provided/Grouped Information (9/46): This category focuses on functionalities aimed at presenting aggregated information related to TD. It includes requirements that point to clustering information to enable decisions, for example, regarding the part of the project that should be the focus of maintenance actions. Grouping information enables stakeholders to gain an understanding of the overall state of TD in the project. Examples of such requirements are related to the amount of code smells ("*The CI-script will include the amount of code smells in a project into the report.*"), business impact, i.e., TD interest (e.g., "*The dedicated technical debt dashboard will present the interest of technical debt*"), and code coverage ("*The dashboard will present test code coverage combined with the amount of issues containing technical debt*"). The category also includes requirements for metrics and trends, such as the evolution of code quality metrics provided by linters (e.g., "*The dashboard will present the evolution of metrics provided by linters.*") In contrast to the detailed information, grouped information requirements help stakeholders to discern software areas that demand attention, prioritize actions based on such areas, and formulate strategies to address TD on those areas, as stated in one of the answers: "*s2_r12: Yes, because the package/file division would help on prioritize the issues on focusing the team efforts in specific parts of the software.*".

Tool usage/Tool execution/Workflow execution (5/46): This category encompasses requirements for executing tools alongside the workflows within a software development process (e.g., continuous integration steps). Essentially, workflow-execution-related requirements focus on: (a) reducing the complexity related to the integration between TDM tools (e.g., SonarQube) and development tools (IDEs) and (b) reducing the disturbance that TDM tools may cause to developers. Essential requirements include optimizing refactoring to avoid bothering developers (e.g., "*The refactoring plugin will provide refactoring suggestion at commit time*"), using existing CI steps as triggers for running analysis and notifications ("*The CI script will send notifications to practitioners after each CI cycle.*"), and triggering notifications based on existing development steps, such as committing code to repositories ("*The code review bot will notify the commit author about code smells in a new code patch.*").

Tool usage/Tool execution/Execution criteria (6/46): This category focuses on configuring tools such as the issue labeling bot and CI-scripts to trigger notifications or actions based on specific criteria. Examples include notifying practitioners when a new issue related to a certain feature/file is open (e.g., "*The issue labeling bot will provide a customization option to configure notifications considering the feature under development*") and the time since an issue was opened (e.g., "*The issue labeling bot will provide a customization option to configure notifications considering the time since an issue was open*"). The requirements in this category indicate that some tools could be configured to execute proactively (i.e., with more automation). This suggests that higher levels of TDM automation are helpful for some tasks (e.g., notifications about TD).

Tool usage/Tool execution/Manual execution (1/46): In addition to automated functionality, practitioners also point to manual execution. This category indicates that practitioners also want to exercise control over when actions are executed within the development process (e.g., "*The refactoring plugin will enable* *manually-triggered execution*"). This manual execution capability provides flexibility and autonomy in managing TD according to practitioners' judgment.

Tool usage/Tool interface (8/46): This category refers to how tools present information to practitioners. Some requirements include communication channels such as e-mail (e.g., "*The issue labeling bot will send notifications by e-mail*") or instant messaging services (e.g., "*The issue labeling bot will send notifications by instant messaging services*"). Another type of interface is a dashboard, which should present a dedicated widget for technical debt, offering practitioners easy access to relevant information (e.g., "*The dashboard will contain a dedicated widget for technical debt*"). Practitioners may also demand a CI Job Dashboard to show the quality metrics collected during CI cycles. IDEs are also appropriate, especially when querying data such as refactoring suggestions (e.g., "*The refactoring plugin will present refactoring suggestions within the IDE interface*"). Overall, this category emphasizes the importance of user-friendly and goal-appropriate interfaces in enhancing the usability and effectiveness of tools within the software development process.

Tool usage/Customization (4/46): Requirements in this category focus on implementing new (customized) tool features. For instance, commands that increase practitioners' control over the tool execution (e.g., "*The issue labeling bot will enable practitioners to choose whether to assign a label to an issue*") or customization options that enable practitioners to consult only relevant data (e.g., "*The dashboard will provide a customization option to select which projects/files to present information about.*").

While the previously presented categories provide an overview of the practitioners' demands, we also investigate the most common requirements. Table 5 shows the top-10 most cited requirements. Eight out of ten are related to information that tools may provide. Among them, the most common requirement is that "*The issue labeling bot will notify practitioners about issues containing technical debt,*" which was reported by 11 respondents in S1. Besides, 7 respondents in S2 stated that a dashboard should include a dedicated widget for technical debt. Overall, 24/46 requirements were supported by at least two respondents, while 22/46 were supported by a single respondent (as we mentioned in Section 3.6, our goal is to report any requirement pointed by practitioners, regardless of the amount of indicators). This information reflects the broader variety of preferences and opinions when it comes to tool usage. Thus, it is fair to argue that highly customizable tools are necessary to suit as many practitioners as possible. Following is our third finding.

Finding #3: Out of 46 requirements, about a half points out to information that the tools may provide, while the other half indicates requirements for using the tools. Among the information to be provided, most requirements indicate detailed information (e.g., lists of code smells). Related to tool usage, requirements for configuring tools' execution are the majority, mostly focusing on criteria for executing tools as well as how this execution happens during the development workflow.

Table 5: Top-10 most cited requirements

Requirement	Category	Resp.
The issue labeling bot will notify practitioners about issues containing technical debt.	information to be provided/detailed information	11
The dashboard will contain a dedicated widget for technical debt	tool usage/tool interface	7
The issue labeling bot will provide a customization option to configure notifications considering the label assigned to an issue.	tool usage/tool execution/execution criteria	6
The issue labeling bot will enable practitioners to choose whether to assign a label to an issue	tool usage/customization	6
The refactoring plugin will enable manually-triggered execution	tool usage/tool execution/manual execution	6
The dashboard will present the time since an issue was opened.	information to be provided/detailed information	5
The ci-script will send notification using instant message services	tool usage/tool interface	5
The dashboard will present a list of smells grouped by package	information to be provided/detailed information	4
The dashboard will present a list of smells grouped by type.	information to be provided/detailed information	4
The dashboard will present the evolution of the metrics provided by linters.	information to be provided/grouped information	4

5 Discussion

5.1 Interpretation of the results of RQ_1

The results of RQ_1 indicate that controlling tool execution is a crucial concern for practitioners. This evidence points to a new perspective not identified in the existing literature about TDM automation, such as the ones presented by Martini et al. (2015) and Biazotto et al. (2023). On the one hand, previous literature and also our study indicate the need for higher levels of automation; on the other hand, our study identified the importance of keeping the practitioners at the center of the automation process. Thus, increasing TDM automation levels must be aligned with human-centered approaches, which enable practitioners to control and monitor tools' execution constantly. Customization options are pragmatic solutions to enhance practitioners' control, since they allow tailoring for specific contexts such as project characteristics and team maturity. Finally, approaches that involve human intervention and customization options may result in enhancing practitioners' trust in tool quality and performance, potentially increasing the usage of tools.

Another finding is that practitioners are highly concerned about the potential disturbance that tools may cause. This disturbance may occur because of excessive information and notification load or because of interfering with development workflows. Previous studies have shown that most TDM tools present interfaces to integrate with IDEs or CI managers (Silva et al., 2022; Biazotto et al., 2023; Avgeriou et al., 2021). The evidence we presented in our study shows a good reason to use those interfaces: practitioners prefer solutions attached to the existing development tools because they cause less disturbance. Regarding the disruption caused by information load, our results point to the need for tools that provide targeted and tailored information instead of many diverse pieces of information that do not add value to practitioners.

5.2 Interpretation of the results of RQ$_2$

In RQ$_2$, we elicited 46 requirements for TD tools, which mainly address the concerns, which are reported in RQ$_1$ by formulating what the tools should offer in a concrete way. In the following paragraphs, we elaborate on how this is done.

First, one of the main goals of using tools for TDM is to gather data to support decision-making about managing TD (e.g., measuring TD or choosing TD items to be paid back). Considering that TD has many facets, ranging from specific problems in source code (e.g., long methods) to architectural problems (e.g., cyclomatic dependencies), many types of data are required to analyze TD accumulation from different perspectives, leading to informed decisions. The need for data is evident in the requirements we collected, with half of them related to the category *Information to be provided.* These requirements address the concerns **Information Load** and **Information Abstraction**, which focus on the amount and type of data provided by tools.

We also noticed that practitioners may resort to executing a tool manually to avoid potential disturbances caused by it and also to check the output of such tools; thus, *Manual execution* shows a pragmatic approach to address the **Human Intervention** concern. Besides, *Tool Execution* accounts for half of the requirements under *Tool Usage*, reflecting the concern **Keep the Control**, i.e., practitioners want to control how and when tools will be executed.

To avoid changes in the development workflow (e.g., waiting for a tool to analyze code in real-time), practitioners prefer to configure when tools should be executed. This is reflected by the identification of several requirements about *Workflow Execution* and *Execution Criteria.* Such requirements are aligned with practitioners' concerns about **Workflow Consistency**.

Finally, several requirements indicate that flexibility is a critical aspect of tool usage. Since software development is a dynamic process in which priorities change frequently, tools for TDM must be able to adapt to those changes. Interestingly, requirements in *all categories* point to tool configuration options, such as choosing data to be presented on dashboards and choosing criteria to execute static code analyzers. Hence, most requirements pointed out by practitioners are related to **Customization**. This emphatically shows that customization options are the main solution for keeping control over tools.

5.3 Implications for practitioners

The model presented in RQ$_1$ outlines common concerns related to adopting, selecting, and using tools. Practitioners can use this model to make informed decisions on how to automate TDM, particularly how to select tools. This is because the model can raise awareness of potential problems involving tools (such as workflow consistency). For instance, instead of focusing only on the features that a tool offers (e.g., identifying TD items), the model helps practitioners to remember that other aspects are also important while choosing tools, such as how easily they can be integrated with the existing development environment (e.g., IDEs). Lastly, the model can facilitate communication among practitioners by defining a common terminology. For instance, the concerns present in the model could be used as categories for documenting tool-related decisions.

The most significant implications of our work are for tool vendors. We provide a model that can help vendors develop tools aligned with practitioners' concerns and a list of requirements that can be directly implemented. In a nutshell, these resources can support the improvement of existing tools and the development of new ones that address developers' concerns and can achieve higher levels of automation and reduced TDM effort.

5.4 Implications for researchers

The results of our study highlight some aspects of TDM and its automation that demand further investigation, providing significant implications for researchers. Firstly, there is a need to explore solutions that address the concerns presented in RQ_1, not only from a technical perspective but also from a sociotechnical one. Our study indicates that while automating TDM is essential for practitioners, maintaining the human element in the automation process is crucial. Therefore, researchers could investigate TDM automation approaches that involve human intervention, for instance, having tools for supporting TD prioritization that receive constant feedback from practitioners, increasing the reliability of the tool (which is also related to the concern **Tool Quality**). Moreover, other non-technical factors, such as team maturity, development processes (e.g., Scrum), and budget, influence tool usage, and a deeper understanding of how these aspects should be addressed while adopting tools is necessary.

From a technical perspective, we identified several requirements related to customizing tool execution (e.g., executing a tool at commit time). Such requirements can potentially increase the level of automation of TDM since tools can operate autonomously once they are configured. Hence, researchers could implement some requirements related to tool execution and conduct empirical studies to evaluate the efficacy of TDM approaches through higher levels of automation.

Researchers can also contribute to improving the accuracy of tools' output; this was identified in RQ_1 as one of the major concerns that practitioners have while adopting tools (i.e., **Tool Quality**). Potential directions include the development of more precise techniques for identifying TD (e.g., based on graph machine learning (Tommasel and Diaz-Pace, 2022)) and improving the estimation of the effort required to repay TD (e.g., combining multiple TD metrics collected from different sources where TD is identified (Tsoukalas et al., 2022)). Furthermore, strategies that combine multiple inputs from different tools and triangulate results (such as the ATD Indexes presented by Verdecchia et al. (2020) and Sas and Avgeriou (2023)) could improve the precision of tools and encourage their adoption by practitioners.

5.5 Relation between practitioners profiles and requirements

Practitioners within a software development team have different profiles (e.g., role, experience, and education), which might impact their style and priorities for managing TD (e.g., checking project overviews rather than specific TD items). In the context of our work, the requirements may be influenced by these profiles. Therefore, we conjectured that the characteristics of practitioners' profile could impact the requirements they have for TDM tools. We verified this hypothesis but did not find strong evidence to support it. For this reason, we do not present this investigation in the Study Design and Results section. Instead, we briefly introduce the steps we followed to verify the hypothesis and a summary of the results in this section. Additionally, all datasets and scripts we used are available in our replication package [7], which might assist other researchers in future work.

First, we envisioned to conduct Chi-Square Tests of Independence (McHugh, 2013). Since the test requires at least five data points per category, we first removed profiles underrepresented in the dataset (e.g., the "Costumer Engineer" role, which had a single respondent). We then ran the tests, but the results were insufficient to reject the null hypothesis. The second approach involved the visual inspection of frequencies plotted using stacked bar charts and 3D scatter plots (i.e., correlating the three profile characteristics within each requirement category). We conducted this analysis using both absolute numbers and normalized percentages, and we observed that the frequencies of respondents per profile category were similar across the requirement categories. Finally, we tried a qualitative approach by reviewing the indicators related to different profiles within requirement categories. However, indicators did not provide new information that could be linked to the profile characteristics.

After we attempted to check the correlation between respondents' profiles and requirements for tools, we concluded that our dataset is not ideal for conducting correlation analysis due to the imbalance in profiles (e.g., more than half of the answers were provided by developers). Nonetheless, we still see value in exploring this hypothesis and plan to do so in future work. Specifically, an interview-based study could provide a more balanced dataset, facilitating the identification of trends. Additionally, an interview-based study allows the collection of more detailed information about practitioners' preferences, which could be linked to their profiles and help to test our hypothesis.

6 Threats Validity

As in any empirical study, there are several threats to the validity of our study. We organize such threats considering the guidelines presented by Wohlin et al. (2012),Guba (1981), and Cruzes and Dyba (2011).

Construct validity (*credibility* in the context of thematic synthesis (Guba, 1981)) regards the connection between the RQs and the study objects. To some extent, the study's findings depend on how the participants were selected and how they interpreted the questions on the questionnaires. Participants recruitment is a potential threat since we rely on convenience sampling. It is possible that the knowledge of the selected group is not representative of the whole population, and the results might change if different participants were selected. To mitigate this threat, we disseminated the survey using social media (e.g., LinkedIn) and also to developers that contribute to OSS projects

[7]The replication package can be accessed on https://zenodo.org/records/13175210

(such as the ones in Apache Software Foundation), trying to reach as many diverse practitioners as possible.

Practitioners' understanding about the scenarios and the questions might impact their answers, thus both scenarios and the questionnaires' design are potential threats in our study. To mitigate this threat, we first based the scenarios on existing literature (such as Martini et al. (2018) and Silva et al. (2022)). In addition, we reviewed the represented technologies (e.g., Grafana Dashboard) to ensure that the scenario is aligned with industrial practice. Finally, we had two rounds of reviews to refine the scenarios (pre-pilot and pilot). Nonetheless, we acknowledge that the results of our study are directly impacted by the scenarios, and results could be different if other scenarios were used.

Finally, since the questions we asked practitioners are largely based on the scenarios, and the scenarios do not cover all possible characteristics of TD management, key concerns not captured in the presented scenarios might be missing. To mitigate this threat, we also included open-ended questions that provided respondents with the opportunity to express their unique views.

External validity (*transferability* in the context of thematic synthesis (Guba, 1981)) is concerned with the degree to which the findings can be generalized from the sample to the population. The non-probabilistic sampling design used for data collection is a potential threat for the external validity of the study. For instance, there is a risk of a biased sample, which is not representative of the target population, or with a dominant participation of a certain practitioner profile (e.g., software developers). To mitigate this threat, the survey was distributed not only through the personal networks of the authors but also through organizations and social media platforms. As a result, the respondents have a fair variety of profiles, as reported in Section 4.1. However, we acknowledge that our findings are deeply bounded to our sample (as any other survey), and the results might not represent the entire population (i.e., software practitioners). Hence, practitioners with roles other than the ones listed on Table 2, such as testers, might have other perceptions about tool usage, which were not captured in our study.

Reliability (*confirmability* and *dependability* in the context of thematic synthesis (Guba, 1981)) considers the bias from the researchers in data collection or data analysis. To mitigate bias from the interpretation of open-ended responses (i.e., reasons), we applied thematic synthesis, which is a well-established method for qualitative data analysis. We also applied an integrated approach for open coding, encompassing three steps. Specifically, the first three authors coded a sample of the answers, discussed potential disagreements, and defined an initial set of codes. Next, the first author and an external collaborator coded all the data. The four coders then checked all the generated codes. In the conflicting cases, the first three authors and the external collaborator discussed the codes until achieving consensus.

For eliciting the requirements, we also relied on a well-established template by Pohl (2016), which helped define a common terminology to document the requirements. Besides that, the second and third authors reviewed the requirements and categories documented by the first author and discussed the disagreements until reaching a consensus.

Finally, although we used a grounded-theory-based approach for coding the answers and defining the themes (as presented in Section 3.5), not all steps of grounded theory were followed. Specifically, data collection and analysis steps were not concurrent (i.e., we collected all the data before analyzing it), which prevented us from claiming the theoretical saturation of our model of concerns. Thus, replications of our study can help capture potential concerns not identified in RQ_1 and move the model toward theoretical saturation. To support such replications and reproductions, we created a replication package [8] with all the necessary data and scripts to run the analyses.

7 Conclusion

The management of TD is crucial for the success of software projects, and using tools that support TDM may reduce the workload of this process through automation. This study surveyed practitioners' perceptions, concerns, and requirements for TDM tools, which can help them use TDM tools more efficiently. In total, we obtained 103 responses (89 valid), from which we proposed a model that presents the main concerns of practitioners regarding TDM tools (as previously shown in Figure 5). Moreover, we elicited 46 requirements that can be used as the basis for improving existing tools or developing new tools. Overall, the results point to the need for highly customizable tools, which can support and balance the potential disturbance that tools can cause to practitioners (e.g., many notifications or changes in the development workflow). Also, customized options can enhance practitioners' control over tools.

As part of our future work, we intend to prioritize the requirements elicited and check which are more relevant to be implemented. In particular, we goal to develop new features within existing tools (such as the Grafana dashboard) to support the prioritized requirements. Besides, we intend to evaluate the model of concerns in real projects, for instance, by interviewing practitioners about past experiences in selecting tools. We can then evaluate the model's effectiveness for supporting decision-making and also capture more concerns not identified. Such evaluation would also support proposing guidelines that can benefit practitioners while using the model, such as decisions trees based on the concerns.

References

Avgeriou, P., Ozkaya, I., Chatzigeorgiou, A., Ciolkowski, M., Ernst, N.A., Koontz, R.J., Poort, E., Shull, F.: Technical debt management: The road ahead for successful software delivery. In: IEEE/ACM International Conference on Software Engineering: Future of Software Engineering (ICSE-FoSE), pp. 15–30 (2023). https://doi.org/10.1109/ICSE-FoSE59343.2023.00007

Apa, C., Solari, M., Vallespir, D., Travassos, G.H.: A taste of the software industry perception of technical debt and its management in uruguay: A survey in software industry. In: 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–9 (2020). https://doi.org/10.1145/3382494.3421463

[8][9]

Avgeriou, P., Taibi, D., Ampatzoglou, A., Fontana, F.A., Besker, T., Chatzigeorgiou, A., Lenarduzzi, V., Martini, A., Moschou, A., Pigazzini, I., Saarimaki, N., Sas, D.D., Toledo, S.S., Tsintzira, A.A.: An overview and comparison of technical debt measurement tools. IEEE Software **38**(3), 61–71 (2021) https://doi.org/10.1109/ms.2020.3024958

Biazotto, J.P., Feitosa, D., Avgeriou, P., Nakagawa, E.Y.: Technical debt management automation: State of the art and future perspectives. Information and Software Technology, 107375 (2023) https://doi.org/10.1016/j.infsof.2023.107375

Besker, T., Martini, A., Bosch, J.: The pricey bill of technical debt: When and by whom will it be paid? In: IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 13–23 (2017). https://doi.org/10.1109/ICSME.2017.42

Besker, T., Martini, A., Bosch, J.: Technical debt cripples software developer productivity: A longitudinal study on developers' daily software development work. In: International Conference on Technical Debt (TechDebt), pp. 105–114 (2018). https://doi.org/10.1145/3194164.3194178

Besker, T., Martini, A., Bosch, J.: Software developer productivity loss due to technical debt—a replication and extension study examining developers' development work. Journal of Systems and Software **156**, 41–61 (2019) https://doi.org/10.1016/j.jss.2019.06.004

Cruzes, D.S., Dyba, T.: Recommended steps for thematic synthesis in software engineering. In: International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 275–284 (2011). https://doi.org/10.1109/ESEM.2011.36

Cunningham, W.: The wycash portfolio management system. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), pp. 29–30 (1992). https://doi.org/10.1145/157709.157715

Codabux, Z., Williams, B.J., Bradshaw, G.L., Cantor, M.: An empirical assessment of technical debt practices in industry. Journal of Software: Evolution and Process **29**(10), 1894 (2017) https://doi.org/10.1002/smr.1894

Digkas, G., Chatzigeorgiou, A., Ampatzoglou, A., Avgeriou, P.: Can clean new code reduce technical debt density? IEEE Transactions on Software Engineering **48**(5), 1705–1721 (2022) https://doi.org/10.1109/TSE.2020.3032557

Ernst, N.A., Bellomo, S., Ozkaya, I., Nord, R.L., Gorton, I.: Measure it? manage it? ignore it? software practitioners and technical debt. In: 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 50–60 (2015). https://doi.org/10.1145/2786805.2786848

Easterbrook, S., Singer, J., Storey, M.-A., Damian, D.: Selecting Empirical Methods for

Software Engineering Research, pp. 285–311. Springer, London, UK (2008). https://doi.org/10.1007/978-1-84800-044-5_11

Fitzgerald, B., Stol, K.-J.: Continuous software engineering: A roadmap and agenda. Journal of Systems and Software **123**, 176–189 (2017) https://doi.org/10.1016/j.jss.2015.06.063

Guba, E.G.: Criteria for assessing the trustworthiness of naturalistic inquiries. ECTJ **29**(2) (1981) https://doi.org/10.1007/bf02766777

Holvitie, J., Leppänen, V., Hyrynsalmi, S.: Technical debt and the effect of agile software development practices on it - An industry practitioner survey. In: 6th International Workshop on Managing Technical Debt (MTB), pp. 35–42 (2014). https://doi.org/10.1109/MTD.2014.8

Jeronimo Junior, H., Travassos, G.H.: Consolidating a common perspective on technical debt and its management through a tertiary study. Information and Software Technology **149**, 106964 (2022) https://doi.org/10.1016/j.infsof.2022.106964

Khomyakov, I., Makhmutov, Z., Mirgalimova, R., Sillitti, A.: Automated measurement of technical debt: A systematic literature review. In: 21st International Conference on Enterprise Information Systems (ICEIS), pp. 95–106 (2019). https://doi.org/10.5220/0007675900950106

Kashiwa, Y., Nishikawa, R., Kamei, Y., Kondo, M., Shihab, E., Sato, R., Ubayashi, N.: An empirical study on self-admitted technical debt in modern code review. Information and Software Technology **146**, 106855 (2022) https://doi.org/10.1016/j.infsof.2022.106855

Kitchenham, B.A., Pfleeger, S.L.: Personal Opinion Surveys, pp. 63–92. Springer, London, UK (2008). https://doi.org/10.1007/978-1-84800-044-5_3

Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software **101**, 193–220 (2015) https://doi.org/10.1016/j.jss.2014.12.027

Li, Y., Soliman, M., Avgeriou, P.: Identifying self-admitted technical debt in issue tracking systems using machine learning. Empirical Software Engineering **27**(6) (2022) https://doi.org/10.1007/s10664-022-10128-3

Linåker, J., Sulaman, S., Host, M., Mello, R.: Guidelines for Conducting Surveys in Software Engineering. Lund University, Lund, Sweden (2015)

Lim, E., Taksande, N., Seaman, C.: A balancing act: What software practitioners have to say about technical debt. IEEE Software **29**(6), 22–27 (2012) https://doi.org/10.1109/MS.2012.130

Martini, A., Besker, T., Bosch, J.: Technical debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations. Science of Computer Programming **163**, 42–61 (2018) https://doi.org/10.1016/j.scico.2018.03.007

Martini, A., Bosch, J., Chaudron, M.: Investigating architectural technical debt accumulation and refactoring over time: A multiple-case study. Information and Software Technology **67**, 237–253 (2015) https://doi.org/10.1016/j.infsof.2015.07.005

McHugh, M.L.: The chi-square test of independence. Biochemia Medica, 143–149 (2013) https://doi.org/10.11613/bm.2013.018

Melo, A., Fagundes, R., Lenarduzzi, V., Santos, W.B.: Identification and measurement of requirements technical debt in software development: A systematic literature review. Journal of Systems and Software **194**, 111483 (2022) https://doi.org/10.1016/j.jss.2022.111483

Malakuti, S., Ostroumov, S.: The quest for introducing technical debt management in a large-scale industrial company. In: 12th European Conference on Software Architecture (ECSA), pp. 296–311 (2020). https://doi.org/10.1007/978-3-030-58923-3_20

Pohl, K.: Requirements Engineering Fundamentals: a Study Guide for the Certified Professional for Requirements Engineering Exam-foundation level-IREB Compliant. Rocky Nook, Inc., Santa Barbara, US (2016)

Rios, N., Mendonça Neto, M.G., Spínola, R.O.: A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology **102**, 117–145 (2018) https://doi.org/10.1016/j.infsof.2018.05.010

Rios, N., Spínola, R.O., Mendonça, M., Seaman, C.: The most common causes and effects of technical debt: First results from a global family of industrial surveys. In: 12th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–10 (2018). https://doi.org/10.1145/3239235.3268917

Rios, N., Spínola, R.O., Mendonça, M., Seaman, C.: The practitioners' point of view on the concept of technical debt and its causes and consequences: a design for a global family of industrial surveys and its first results from Brazil. Empirical Software Engineering **25**(5), 3216–3287 (2020) https://doi.org/10.1007/s10664-020-09832-9

Sas, D., Avgeriou, P.: An architectural technical debt index based on machine learning and architectural smells. IEEE Transactions on Software Engineering **49**(8), 4169–4195 (2023) https://doi.org/10.1109/TSE.2023.3286179

Strauss, A., Corbin, J.: Basics of Qualitative Research: Grounded Theory Procedures and Techniques. Sage Publications, Thousand Oaks, US (1990)

Silva, V.M., Junior, H.J., Travassos, G.H.: A taste of the software industry perception of technical debt and its management in Brazil. Journal of Software Engineering Research and Development **7**, 1–16 (2019) https://doi.org/10.5753/jserd.2019.19

Silva, J.D.S., Neto, J.G., Kulesza, U., Freitas, G., Rebouças, R., Coelho, R.: Exploring technical debt tools: A systematic mapping study. In: International Conference on Enterprise Information Systems (ICEIS), pp. 280–303 (2022). https://doi.org/10.1007/978-3-031-08965-7_14

Sas, D., Pigazzini, I., Avgeriou, P., Fontana, F.A.: The perception of architectural smells in industrial practice. IEEE Software **38**(6), 35–41 (2021) https://doi.org/10.1109/MS.2021.3103664

Tommasel, A., Diaz-Pace, J.A.: Identifying emerging smells in software designs based on predicting package dependencies. Engineering Applications of Artificial Intelligence **115**, 105209 (2022) https://doi.org/10.1016/j.engappai.2022.105209

Tsoukalas, D., Mittas, N., Chatzigeorgiou, A., Kehagias, D., Ampatzoglou, A., Amanatidis, T., Angelis, L.: Machine learning for technical debt identification. IEEE Transactions on Software Engineering **48**(12), 4892–4906 (2022) https://doi.org/10.1109/TSE.2021.3129355

Verdecchia, R., Lago, P., Malavolta, I., Ozkaya, I.: ATDx: Building an architectural technical debt index. In: 15th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE), pp. 1–8 (2020). https://doi.org/10.5220/0009577805310539

Solingen, R., Basili, V., Caldiera, G., Rombach, H.D.: Goal Question Metric (GQM) Approach. John Wiley & Sons, Inc. (2002). https://doi.org/10.1002/0471028959.sof142

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A.: Experimentation in Software Engineering. Springer, Heidelberg, Germany (2012). https://doi.org/10.1007/978-3-642-29044-2

Zazworka, N., Vetro', A., Izurieta, C., Wong, S., Cai, Y., Seaman, C., Shull, F.: Comparing four approaches for technical debt identification. Software Quality Journal **22**(3), 403–426 (2013) https://doi.org/10.1007/s11219-013-9200-8