# Asking about Technical Debt: Characteristics and Automatic Identification of Technical Debt Questions on Stack Overflow

Nicholas Kozanidis[*]
Vrije Universiteit Amsterdam
The Netherlands
nickkozanidis@gmail.com

Roberto Verdecchia[*] [†]
Vrije Universiteit Amsterdam
The Netherlands
r.verdecchia@vu.nl

Emitzá Guzmán
Vrije Universiteit Amsterdam
The Netherlands
e.guzmanortega@vu.nl

## ABSTRACT

**Background:** Q&A sites allow to study how users reference and request support on technical debt. To date only few studies, focusing on narrow aspects, investigate technical debt on Stack Overflow.

**Aims:** We aim at gaining an in-depth understanding on the characteristics of technical debt questions on Stack Overflow. In addition, we assess if identification strategies based on machine learning can be used to automatically identify and classify technical debt questions.

**Method:** We use automated and manual processes to identify technical debt questions on Stack Overflow. The final set of 415 questions is analyzed to study (i) technical debt types, (ii) question length, (iii) perceived urgency, (iv) sentiment, and (v) themes. Natural language processing and machine learning techniques are used to assess if questions can be identified and classified automatically.

**Results:** Architecture debt is the most recurring debt type, followed by code and design debt. Most questions display mild urgency, with frequency of higher urgency steadily declining as urgency rises. Question length varies across debt types. Sentiment is mostly neutral. 29 recurrent themes emerge. Machine learning can be used to identify technical debt questions and binary urgency, but not debt types.

**Conclusions:** Different patterns emerge from the analysis of technical debt questions on Stack Overflow. The results provide further insights on the phenomenon, and support the adoption of a more comprehensive strategy to identify technical debt questions.

## CCS CONCEPTS

• **Software and its engineering → Collaboration in software development**; **Software evolution**; **Maintaining software**; • **Computing methodologies** → *Supervised learning*.

**ACM Reference Format:**
Nicholas Kozanidis, Roberto Verdecchia, and Emitzá Guzmán. 2022. Asking about Technical Debt: Characteristics and Automatic Identification of Technical Debt Questions on Stack Overflow. In *ACM / IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM) (ESEM '22), September 19–23, 2022, Helsinki, Finland.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3544902.3546245

[*]The first two authors contributed equally to this work.
[†]Corresponding author.

## 1 INTRODUCTION

During the past years, a plethora of different research methods and data sources have been used to study Technical Debt (TD). Among these studies, quantitative investigations relying on source code analysis were frequently used to identify and manage TD [7, 15, 21, 35, 41, 61]. Researchers also used qualitative approaches, such as surveys, focus groups, and grounded theory, to acquire knowledge from those directly experiencing TD and gain insights that could be missed by utilizing exclusively quantitative approaches [25, 32, 34, 49, 59]. Among the various data sources used, a vast pool of data which may contain information useful to further understand TD remains almost uncharted. With over 23M questions asked, 18M registered users, and 11M visits per day[1], Stack Overflow (SO) ranks top in all these metrics among the popular umbrella of Stack Exchange Question and Answer (Q&A) sites. Possibly, SO is to date largest and most popular Q&A site where users ask questions, share knowledge, and discuss topics related to computer programming. Despite its large popularity, relatively few research has investigated TD from the point of view of SO.

In this study, we aim at understanding the nature of questions regarding technical debt posted on Stack Overflow, which from now on are referred to as Technical Debt Questions (TDQs). Specifically, we aim at characterizing the TD phenomenon through the lens of SO by studying different characteristics of TDQs, such as TD types, question length, urgency, sentiments, and considered themes. In addition, since we have to rely on a keyword-based search followed by manual analysis to identify TDQs, we also investigate the potential of predictive models for automatically identifying and categorizing TDQs.

With regards to scientific progress, our study constitutes a first systematic step towards understanding how users reference and request support on TD online, by providing a combination of quantitative and qualitative empirical results, a ready-to-be-used labeled dataset, and predictive models to build on our findings. Furthermore, our study supports practitioners in understanding how Stack Overflow is used to ask TDQs, and access structured knowledge which could not be gained by considering their personal experience alone.

The main contributions of this study are the following:

- An in-depth analysis of technical debt questions on Stack Overflow, including discussed TD types, question length, urgency, sentiments, and themes considered;
- A fine-grained dataset of labelled technical debt Stack Overflow questions;
- Predictive models to automatically detect and classify technical debt questions on Stack Overflow.

The replication package of this study is available at the following online repository: `https://github.com/TD-SO/rep-pkg`.

[1]https://stackexchange.com/sites#traffic. Accessed 28th April 2022.

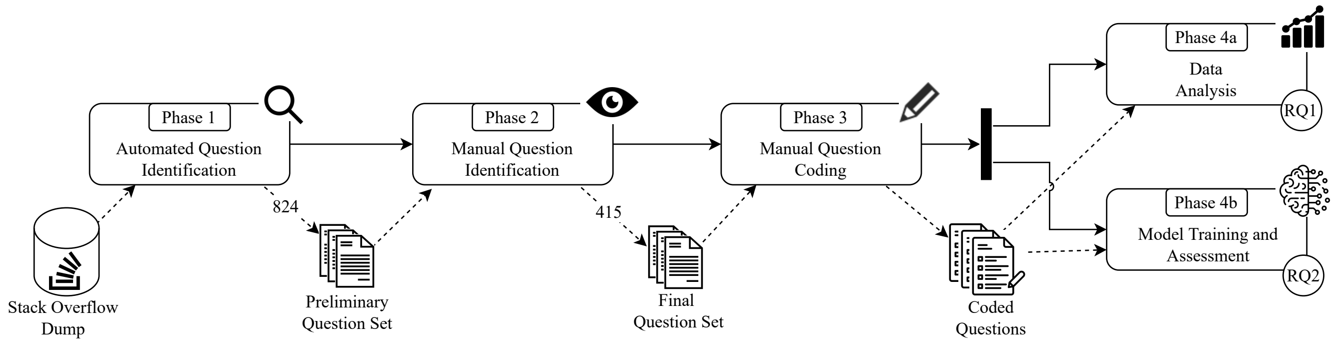


**Figure 1: Research process overview.**

## 2 STUDY DESIGN AND EXECUTION

In this section, we document the design and execution of this study, in terms of research goal (Section 2.1), research questions (Section 2.2), and research process followed (Section 2.3).

### 2.1 Research Goal

The goal of this research is twofold. On one hand, we aim to broaden the body of knowledge of TD by studying the characteristics of SO questions posed on the topic. On the other hand, we aim to evaluate if SO questions related to TD can be identified and classified automatically *via* predictive models. More formally, by utilizing the Goal-Question-Metric approach [8], the objective of this research can be formulated as follows:

***Analyze*** *the content of questions*
***For the purpose of*** *characterization and automatic classification*
***With respect to*** *technical debt*
***From the viewpoint of*** *software engineering researchers*
***In the context of*** *Stack Overflow.*

### 2.2 Research Questions

Based on our goal, we can derive the following two research questions (RQs), which are at the foundation of our study:

**$\text{RQ}_1$:** *What are the characteristics of technical debt questions on Stack Overflow?*

With $RQ_1$, we aim at gaining a better understanding of the TD phenomenon, by studying the topic through the lens of questions posed on the SO platform. Different aspects, namely TD types, question length, urgency, sentiments, and emerging themes are considered to answer this RQ (as further detailed in Section 2.3).

**$\text{RQ}_2$:** *Can technical debt questions on Stack Overflow be automatically identified and classified via predictive models?*

With $RQ_2$, we aim to evaluate if TDQs can be automatically identified and classified *via* predictive models. Positively answering this RQ allows to strive away from trivial keyword-based SO post search strategies (such as the one used in the first phase of this study, see Section 2.3.1), and enables to identify posts related to TD that do not explicitly mention the term "technical debt" or similar keywords. Also, it reduces the manual analysis entailed in TD categorization.

### 2.3 Research Process

An overview of the research process we follow to answer our RQs is depicted in Figure 1, and is further detailed in the reminder of this section. Our research process comprises five phases. In the first phase (Phase 1, Section 2.3.1), we identify a preliminary set of TDQs *via* an automated search process. Subsequently, we refine the TDQ set *via* a manual process (Phase 2, Section 2.3.2). The final TDQ set is then manually coded according to different characteristics related to TD (Phase 3, Section 2.3.3). In the last two phases, we analyze the coded data to answer $RQ_1$ (Phase 4a, Section 2.3.4), and execute a predictive model training and assessment to answer $RQ_2$ (Phase 4b, Section 2.3.4).

#### 2.3.1 *Phase 1: Automated Question Identification.*

In our first research phase, we identify a preliminary TDQ set by executing an automated query on the *Stack Exchange Data Dump*[2]. Specifically, we select for inclusion questions posted on SO which contain in their title, body, and/or tags, at least one of the following keywords: "`technical debt`", `''technical debts''`, `''technical-debt''`, `''tech debt''`, `''tech-debt''`, and `''techdebt''`. To enrich our automated selection, we include in the preliminary TDQ set also the questions which contain one or more of the aforementioned keywords in at least one of their answers. The automated selection phase terminates with the identification of a preliminary set of 824 TDQs.

#### 2.3.2 *Phase 2: Manual Question Identification.*

In order to ensure the quality of TDQs identified automatically, in this second phase we execute a manual scrutiny of the preliminary TDQ set. Specifically, two researchers manually inspect a distinct subsample of 412 TDQs each (824 in total), and evaluate the content of each question against the 16162 Technical Debt definition [6]. If a question regards TD (as defined in the 16162 TD definition), then it is included in our final TDQ dataset, otherwise it is excluded. Questions concerning the troubleshooting of tools implementing technical debt analyses, *e.g.,* SonarQube[3] installation issues, are excluded, as these questions concern solely technical issues related to tools, and do not contribute to answer our RQs.

The inter-rater reliability of this process is assessed by means of Cohen's Kappa [18] on a set of 50 questions sampled uniformly at random, showcasing a strong agreement among raters (kappa=0.83). The manual identification phase terminates with the

[2]https://archive.org/details/stackexchange. Accessed 21st April 2022.
[3]https://www.sonarqube.org. Accessed 25th April 2022.

identification of approximately half of preliminary set of questions as TD-related (415/824). These 415 TDQs constitute the final TDQ set used in this investigation[4]. Out of the 409 questions not related to TD, a notable portion (174/409) regards technical troubleshooting questions of the SonarQube tool (see Figure 9).

### 2.3.3 *Phase 3: Manual Question Coding.*

In this phase, the final TDQ set is qualitatively analyzed *via* two coding strategies, which are used to derive the *TD type* and *perceived urgency* of the questions. Regarding the coding of TD types, we use provisional coding [51] to map each TDQ to one of the ten TD types presented in the TD type taxonomy presented by Li *et al.* [37]. Regarding perceived urgency, we use magnitude coding [51] to classify the urgency of TDQs as either "none", "very mild", "mild", "moderate", "severe", or "very severe". As for the manual identification of TDQs (cf. Section 2.3.2), two researchers code a distinct subsample of 206 TDQs each. A coding guide, made available in the replication package, is used to ensure both researchers share a common interpretation of the TD types and perceived urgencies.[5] Inter-rater reliability is calculated by means of Cohen's kappa [18] on a sample of 50 TDQs sampled uniformly at random. The inter-rater reliability is strong for TD types (kappa=0.80) and substantial for the perceived urgency (kappa=0.71).

### 2.3.4 *Phase 4a: Data Analysis.*

To answer $RQ_1$, we further analyze the coded data both by quantitative and qualitative means. From a quantitative standpoint, we analyze our TDQs to understand the recurrence of TD types, the perceived urgencies, and the distribution of urgencies across TD types. In addition, we consider the number of words used in each question to investigate if TD types influence TDQs length. To evaluate the affective states expressed in the TDQs, we analyze the questions *via* sentiment analysis [40]. The sentiment analysis is conducted by using the VADER (Valence Aware Dictionary and sEntiment Reasoner) tool[6]. We use VADER as it is a state-of-the-art sentiment analysis tool, and has a high accuracy when applied in the context of SO [38]. The sentiment expressed in the TDQs body is analyzed both in terms of relative sentiment components (negative, neutral, positive) and compound sentiment scores [30]. For this computation, we use the out of the box configuration of VADER.

Regarding the qualitative analysis conducted to answer $RQ_1$, we further codify the content of our 415 TDQs. The goal of this additional coding process is to gain further insights into the content of the TDQs, in terms of considered themes related to TD. A natural coding strategy to achieve this goal would be to use provisional coding [51] by considering as "start list" the sub-categories of TD types presented in the taxonomy of Li *et al.* [37]. Nevertheless, a preliminary data analysis shows that the sub-categories of Li *et al.* [37] are only seldom mentioned in our TDQs.[7] As provisional coding is not a viable option, we analyze the TDQs *via* open and subsequent focused coding [51], leading to the identification of the most salient themes related to TD emerging in our TDQ corpus. The coding process is conducted by a single researcher with six years of experience in TD investigation. To guarantee the quality of the coding process, the final coded data is examined by a second researcher. Potential inaccuracies are jointly scrutinized and revised among the two researchers involved in this phase.

[4]Except for the binary TDQs classifier (see Section 2.3.5), which uses the original 824 set.
[5]For concrete coding examples of TD types, urgency, and theme refer to the coding guide.
[6]https://github.com/cjhutto/vaderSentiment. Accessed 21 April 2022.
[7]A further consideration on this research-related finding is discussed in Section 4.

### 2.3.5 *Phase 4b: Model Training and Assessment.*

To answer $RQ_2$ we develop three types of predictive models[8]: (i) a binary classifier for detecting TDQs, (ii) a multi-class classifier for detecting the type of technical debt in TDQs, (iii) a multi-class classifier for categorizing the urgency of TDQs according to the six-level scale presented in Section 2.3.2, (iv) a binary classifier for categorizing TDQs as either non-urgent (grouping together "none", "very mild", "mild" urgency) or urgent (grouping together "moderate", "severe", and "very severe" urgency). We develop the models using both title and body of the TDQs.

We train and test the binary classifier detecting TDQs with the original SO questions collected in Phase 1. This dataset consists of 824 SO questions (see Section 2.3.1). We train and test the rest of the classifiers with only the questions that were manually labelled as TDQs. This latter dataset consists of 415 TDQs (see Section 2.3.2).

To prepare the text for the classifiers we tokenize all text, convert it into lower case, extract bigrams, remove stopwords and all punctuation marks. We also remove any SO related HTML or markdown elements and replace any appearances of source code with a code tag, since these do not contain information of lexical or semantic importance. Finally, we stem the text, eliminating inflectional forms of words. To avoid over-fitting the models during the training phase we remove the mention of `"technical debt"`, technical debt related keywords (see Section 2.3.1) and `"SonarQube"` from the text.

To train each classifier we convert the pre-processed single tokens and bigrams into a vector space model using tf-idf [47] as a weighting scheme. Besides the vector weights, we input a set of additional features into the model:

- Distinct number of bigrams: Distinct number of bigrams that we find to be highly correlated with posts discussing TD. These bigrams are found within the labelled data set and are scored using the built-in function of the nltk[9] collocations library;
- Distinct number of unigrams: Unigrams acquired similarly to the bigrams.
- Popularity score: The amount of upvotes minus the amount of downvotes a post has on SO;
- Comment count: The number of comments of a SO post;
- Tf-idf score: The summation of tf-idf scores of each term within the body of each post.

All predictive models are built using the scikit-learn[10] library in Python. We train and test the classifiers with a 10-fold cross validation on the previously described labelled dataset. We report our results using standard metrics for machine learning models: precision, recall and $F_1$ score.

We use random forest (RF) models for this phase as an initial empirical evaluation including other machine learning models (SVM, Multinomial Naive Bayes, Logistic Regression and Decision Trees), yielded the best results for these.

To handle the data imbalance in our multi-class classifiers, we use SMOTE [16] as a balancing technique. SMOTE oversamples minority classes by creating synthetic data points. During each fold of the cross validation, the minority classes of the training dataset are oversampled using SMOTE.

[8]We use the terms models and classifiers interchangeably in this work.
[9]https://www.nltk.org. Accessed 27th April 2022.
[10]https://scikit-learn.org. Accessed 27th April 2022.

## 3 RESULTS

In this section, we present the results collected to answer our RQs. The results gathered for $RQ_1$ are documented in Section 3.1, while Section 3.2 is dedicated to the results of $RQ_2$. The interpretation and discussion of the results are presented in Section 4.

### 3.1 Results RQ$_1$: Characteristics of Technical Debt Questions on Stack Overflow

The results of $RQ_1$, dedicated to the characterization of TDQs, are structured as follows. Section 3.1.1 documents the distribution of TDQs across TD types. Section 3.1.2 discusses the lenght of TDQs. The perceived urgency of questions is reported in Section 3.1.3, while the sentiment in Section 3.1.4. Finally, in Section 3.1.5 we detail the themes recurring in the TDQs.

*3.1.1 TDQs Technical Debt Types.* An overview of the distribution of TDQs among TD types, according to the TD taxonomy of Li *et al.* [37], is depicted in Figure 2. Architecture TD is the most recurring TD type (96/415), followed by Code TD (83/415) and Design TD (80/415). Other TD types are less recurring, while Defect TD [37] is never mentioned.

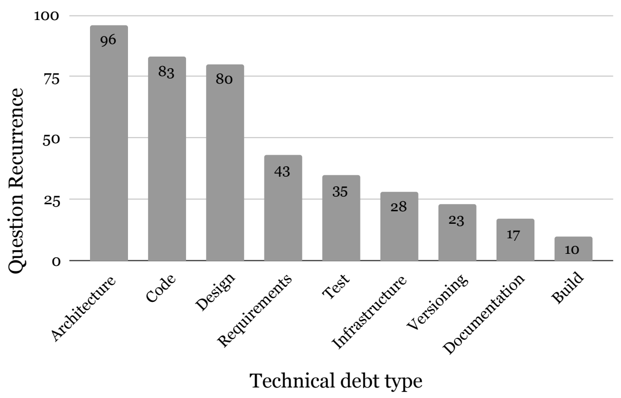


**Figure 2: Technical debt types considered across questions.**

*3.1.2 TDQs Length.* The distribution of TDQs length, grouped by TD types, is documented in Figure 3, while Table 1 shows the related summary statistics. From the values collected we observe that most TD types display comparable question lengths, with ranges which generally lay between 100 and 200 words per question. Median lengths are also comparable across TD types, with median values within the range of 130 and 147 words, with the exception of Design and Test TD, which showcase longer median lengths, and Documentation TD, which is characterized by a shorter median length.

*3.1.3 TDQs Urgency.* The distribution of perceived urgencies across TDQs is documented in Figure 4. In most cases questions present some kind of urgency, with only 29/415 questions displaying none. In most of the cases however, such urgency is only of very mild (154/415) or mild (100/415) nature. Overall, the frequency of questions reporting a higher urgency results to steadily decline as urgency rises, with only a minor portion of questions reporting very severe urgency (13/415).

**Table 1: Summary statistics of question length (in number of words) across technical debt types.**

| TD Type | Min. | Max. | Median | $\sigma$ |
|---|---|---|---|---|
| Architecture | 34 | 679 | 147 | 129.9 |
| Build | 35 | 297 | 132 | 55.73 |
| Code | 30 | 848 | 143 | 146.29 |
| Design | 42 | 774 | 174 | 160.42 |
| Documentation | 30 | 323 | 98 | 77.44 |
| Infrastructure | 42 | 351 | 136 | 74.79 |
| Requirements | 20 | 652 | 130 | 137.38 |
| Test | 44 | 1554 | 178 | 249.07 |
| Versioning | 55 | 339 | 141 | 68.46 |
| Average | 36.88 | 646.33 | 142.11 | 122.164 |

$\sigma$: Standard deviation

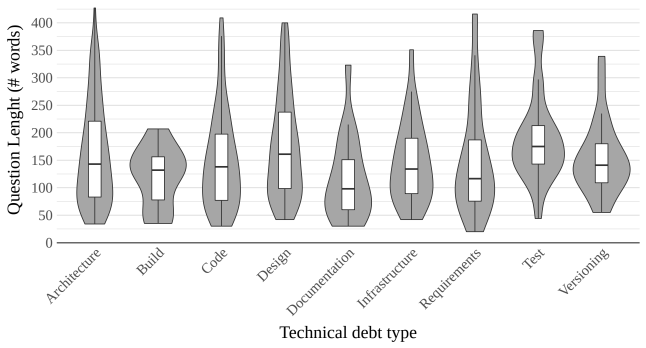


**Figure 3: Question length (in number of words) across technical debt types.**[11]

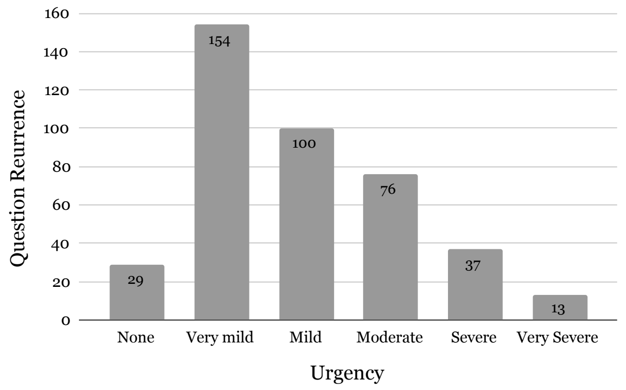
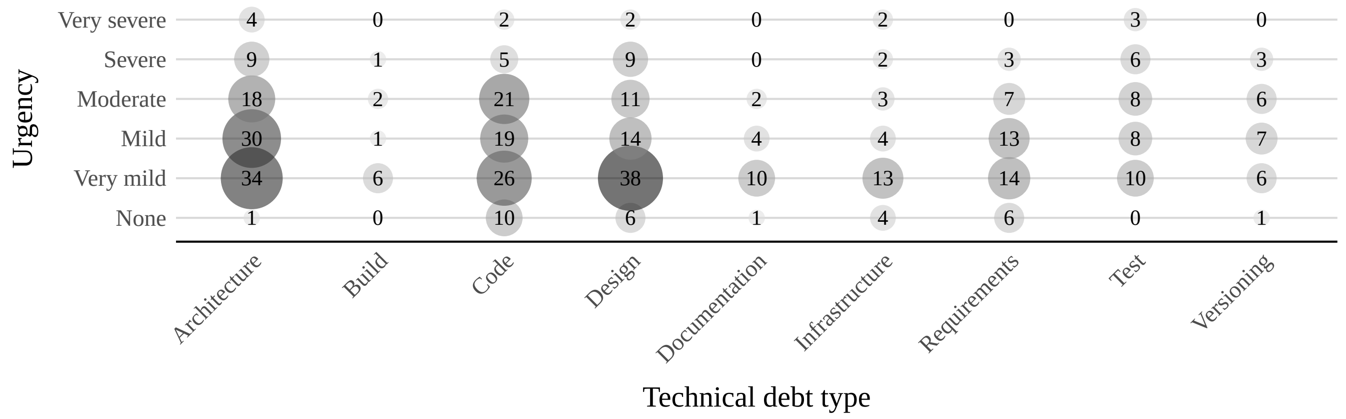


**Figure 4: Urgency expressed in the questions.**

To gain further insights into the perceived urgency expressed in the TDQs, we analyze how urgency levels are distributed across TD types (see Figure 5). By considering relative values, we observe that Test TD is the TD type characterized by the highest urgency, with 9/35 questions expressing severe to very severe urgency. Code and Versioning TD are instead the types reporting the relatively highest moderate urgency across TD types, with Code TD reporting 21/83

[11]For the sake of readability, outlier values are not represented in Figure 3.



| Urgency | Architecture | Build | Code | Design | Documentation | Infrastructure | Requirements | Test | Versioning |
|---|---|---|---|---|---|---|---|---|---|
| Very severe | 4 | 0 | 2 | 2 | 0 | 2 | 0 | 3 | 0 |
| Severe | 9 | 1 | 5 | 9 | 0 | 2 | 3 | 6 | 3 |
| Moderate | 18 | 2 | 21 | 11 | 2 | 3 | 7 | 8 | 6 |
| Mild | 30 | 1 | 19 | 14 | 4 | 4 | 13 | 8 | 7 |
| Very mild | 34 | 6 | 26 | 38 | 10 | 13 | 14 | 10 | 6 |
| None | 1 | 0 | 10 | 6 | 1 | 4 | 6 | 0 | 1 |

**Figure 5: Question urgency distribution per technical debt type across questions.**

moderate urgency questions, and Versioning TD 6/23. Very mild urgency instead results to be more recurrent, in relative terms, in questions regarding Infrastructure TD (13/28), Documentation TD (10/17), and Build TD (6/10). Questions with no urgency are instead less recurrent across all TD types, reflecting the overall urgency trend already observed in Figure 4.

*3.1.4 TDQs Sentiment.* Regarding the sentiment expressed in TDQs, an overview of the distribution of sentiment scores across negative, neutral, and positive sentiment components is documented in Figure 6. All TDQs of our corpus showcase a very noticeable neutral sentiment, with a median neutral sentiment score equal to 0.87. Positive affective states are instead expressed to a much lower extent in the TDQs (median sentiment score = 0.07), and negative sentiment even less (median sentiment score = 0.04). By considering instead the compound sentiment expressed in the TDQs (see Figure 7), we can observe that questions leaning towards an overall more positive sentiment are more recurrent, with a median compound sentiment value of 0.61.

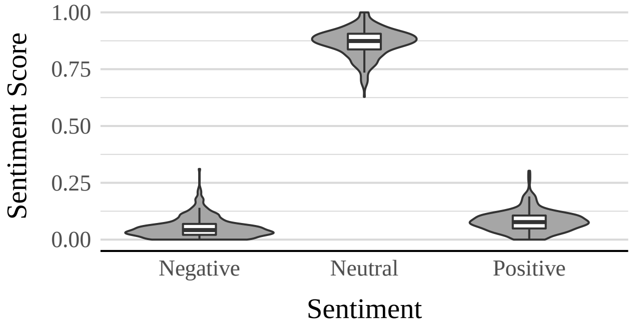


**Figure 6: Sentiment used in technical debt related questions.**

*3.1.5 TDQs Themes.* As final characteristic considered to answer $RQ_1$, we investigate the most salient themes related to TD which emerge from our TDQ corpus (see also Section 2.3.4). Figure 8

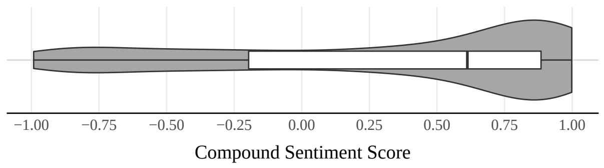


**Figure 7: Compound sentiment distribution across questions.**

shows an overview of all treated themes, while Figure 9 presents the recurrence of the themes (right-most column), and their distribution across TD types (central column). Due to space limitations, themes which are only seldom mentioned in our TDQs corpus are not depicted in Figure 9 and are not further discussed in the reminder of this section. For completeness, such themes are *reverse engineering* (4/415), *dead code* (4/415), *deployment* (4/415), *lack of documentation* (4/415), *missing tests* (4/415), *code churn* (3/415), *unfit build processes* (3/415), and *self-admitted TD* (2/415).

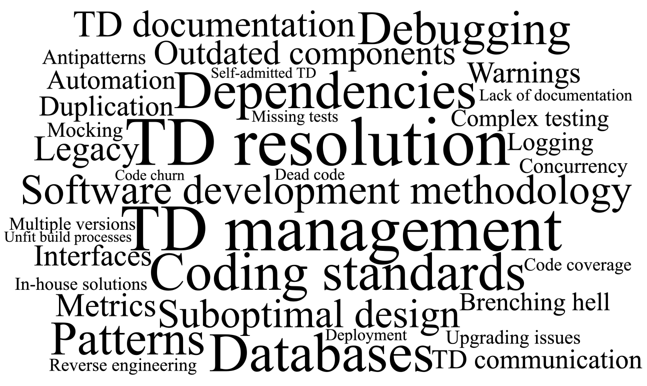


**Figure 8: Themes considered in the questions.**

The overall most recurrent TD theme is *TD resolution* (68/415), which is associated to questions asking, at various levels of abstraction, how to resolve TD items. TD resolution is mentioned

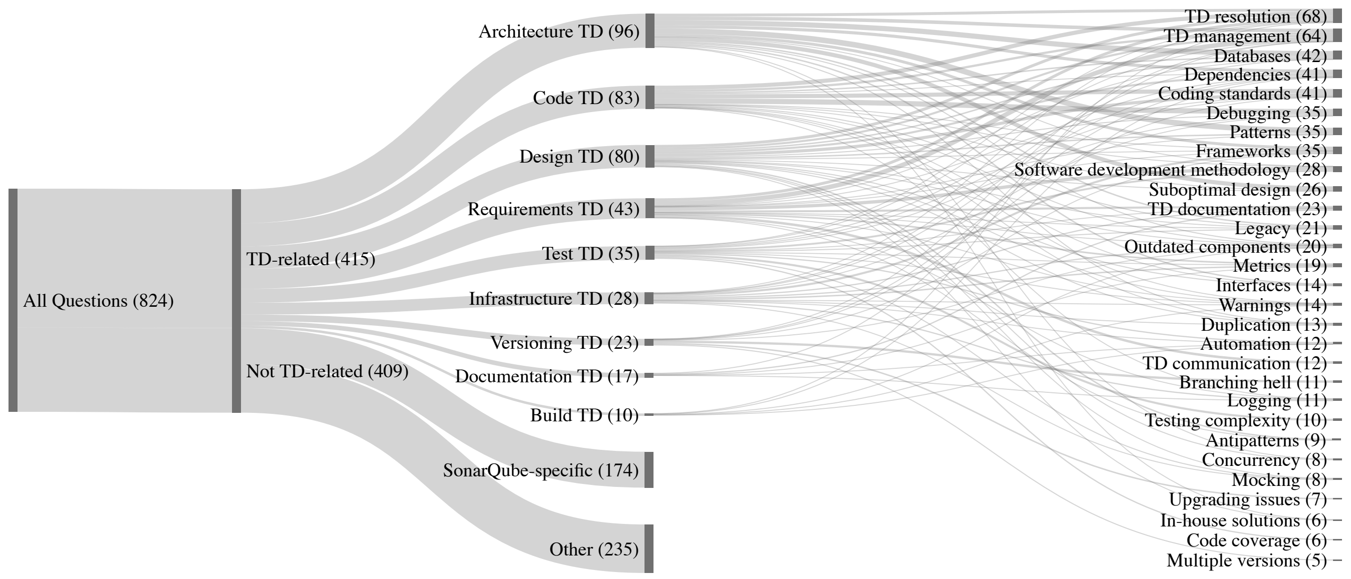


**Figure 9: Distribution of technical debt types and themes considered in the questions.**

to various extents in TDQs regarding almost every TD type, except in Build and Test TD questions.

The second most recurrent theme is *TD management* (64/415), *i.e.,* questions regarding how to properly control and deal with TD. TD management is the most recurring in questions related to Requirements TD (22/43), followed by Architecture (17/96) and Code TD (11/83). TD items related to *databases* are also frequently mentioned (42/415), and mostly occur in questions regarding Architecture TD (25/96), followed by Code (7/83) and Design TD (7/80).

*Dependencies* of software components (at various levels of abstraction), and their relation to TD, are discussed among numerous TD types, such as Architecture (15/96), Design (10/80), Test (5/35), Code (4/83), and Build TD (3/10). *Coding standards* (41/415), *i.e.,* conventions and guidelines used to improve software quality, are instead most mentioned in questions regarding Code (20/83) and Design TD (15/80), followed by few questions on Test (3/35) and Requirements TD (2/43).

*Debugging* (35/415), *i.e.,* questions regarding technical issues related to the resolution of TD items, are most frequent in Code TD questions (25/83), and appear only to a much lower extent in questions regarding other TD types, *e.g.,* Design (5/80) and Test TD (4/35). TD items related to software *patterns* (35/415) are instead almost exclusively mentioned in questions regarding Architecture (27/96) and Design TD (7/80).

TDQs considering *software frameworks* (35/415), *e.g.,* Spring Boot[12] and .NET[13], are most frequently mentioned in the context of Architecture TD (14/96), while being quite frequent in Infrastructure TD questions (10/28). In contrast, TDQs considering *software development methodologies* (28/415), *e.g.,* Agile and test driven development, appear almost exclusively in questions regarding Requirements (16/43) and Test TD (9/35).

*Sub-optimal design* (26/415) is mostly referenced in the context of Architecture TD (20/96), while the documentation of TD items (*TD documentation*, 23/415) is most occurring in Requirements (10/43), Infrastructure (6/28), and Documentation TD (6/17) questions.[14]

*Legacy* code (21/415) is mentioned most frequently in the context of Architecture (7/96), Code (5/83), and Test TD (5/35). TD items related to *outdated components* (20/415) instead occur most frequently in questions on Infrastructure (6/28) and Architecture TD (5/96).

Software *metrics* (19/415) are discussed in questions regarding Requirements (8/43) and Infrastructure TD (7/28), while TD items related to *interfaces* (14/415) are almost exclusively mentioned in Architecture (7/96) and Design TD questions (5/80).

Some of the themes result to be exclusive only to certain TD types. For example, TD incurred due to *upgrading issues* (7/415), utilizing *multiple versions* of a product (7/415), or using multiple concurrent development branches (*branching hell*, 11/415), are all mentioned exclusively in the context of Versioning TD.

Similarly, TD related to *mocking* practices (8/415), *automation* of development processes (12/415), *testing complexity* (10/415), and *code coverage* (6/415) is mentioned mostly if not exclusively in Test TDQs.

TD arising in the context of *logging* (11/145) is most mentioned in questions related to Infrastructure TD (6/28). In contrast, TD related to *warnings* (14/415) is referenced to small extents in different TD types, *e.g.,* Code (3/83), Test (3/35), and Build TD (3/10).

Some rather infrequent themes, namely TD regarding code *duplication* (13/415), *concurrency* (8/415), *antipatterns* (9/415), or using solutions developed in-house instead of already available ones (*in-house solutions*, 6/415), mostly occur in Design TD questions.

Finally, the communication of concepts related to TD among colleagues and/or stakeholders of a software product (*TD*

[12]https://spring.io/projects/spring-boot. Accessed 22nd April 2022.
[13]https://dotnet.microsoft.com/en-us. Accessed 22nd April 2022

[14]"Documentation TD" regards insufficient, incomplete, or outdated documentation [37]. "TD documentation" instead refers to *the documentation of* TD.

*communication*, 12/415) is a theme that emerges exclusively in questions regarding Requirements TD (12/43).

## 3.2 Results RQ$_2$: Automatic Classification of Technical Debt Questions on Stack Overflow

The binary classifier is able to detect TDQs and non-TDQs with a very good precision and recall, with an average of 0.81 and 0.80, respectively (see Table 2).

**Table 2: Results for binary classifier detecting Technical Debt.**

| | Precision | Recall | F$_1$ score |
|---|---|---|---|
| Non-TDR | 0.80 | 0.76 | 0.78 |
| TDR | 0.81 | 0.85 | 0.83 |
| Average | 0.81 | 0.80 | 0.80 |

**Table 3: Results for multi-class classifier categorizing types of Technical Debt.**

| TD Type | Precision | Recall | F$_1$ score |
|---|---|---|---|
| Requirements | 0.43 | 0.53 | 0.47 |
| Infrastructure | 0.42 | 0.55 | 0.48 |
| Code | 0.28 | 0.23 | 0.25 |
| Testing | 0.58 | 0.75 | 0.66 |
| Architecture | 0.54 | 0.33 | 0.41 |
| Design | 0.36 | 0.36 | 0.36 |
| Versioning | 0.49 | 0.5 | 0.49 |
| Documentation | 0.65 | 0.57 | 0.61 |
| Build | 0.69 | 0.68 | 0.68 |
| Average | 0.49 | 0.50 | 0.49 |

Results are less promising for the multi-class classifiers categorizing TD types. The multi-class classifier for TD types has an average precision of 0.49 and recall of 0.50 (see Table 3). This classifier performs best when classifying the Test (precision=0.58, recall=0.75) and Build (precision=0.69, recall=0.68) TD types. The least performing types are Design (precision=0.36, recall=0.36) and Code TD (precision=0.28, recall=0.23).

The urgency multi-class classifier has an average precision and recall of 0.64 for both metrics (see Table 4). When training a model that categorizes the TDR posts into urgent and non-urgent, we obtain an average precision and recall of 0.84 (see Table 5).

**Table 4: Results for multi-class classifier categorizing types of urgency in Technical Debt related questions.**

| Urgency (six-level scale) | Precision | Recall | F$_1$ score |
|---|---|---|---|
| None | 0.82 | 0.7 | 0.75 |
| Very mild | 0.41 | 0.35 | 0.38 |
| Mild | 0.6 | 0.5 | 0.55 |
| Moderate | 0.5 | 0.73 | 0.59 |
| Severe | 0.64 | 0.71 | 0.68 |
| Very severe | 0.88 | 0.82 | 0.85 |
| Average | 0.64 | 0.64 | 0.63 |

**Table 5: Results for binary classifier categorizing urgency in Technical Debt related questions.**

| Urgency (binary) | Precision | Recall | F$_1$ score |
|---|---|---|---|
| Non-urgent | 0.85 | 0.81 | 0.83 |
| Urgent | 0.82 | 0.87 | 0.85 |
| Average | 0.84 | 0.84 | 0.84 |

## 4 DISCUSSION

Almost half of the questions collected *via* our automated search query (see Section 2.3.2) are manually identified as non-TD-related (409/824), with a notable portion concerning the SonarQube tool (174/824). The high occurrence of non-TD-related questions is mostly caused by irrelevant mentions of the term "technical debt" (frequently occurring in low-rated answers), and the presence of the term in code/log snippets. The high number of questions related to SonarQube troubleshooting (see Section 2.3.2) is instead attributable to the potentially high popularity of the tool to identify code-related TD.

Regarding the characteristics of TDQs ($RQ_1$), a first consideration can be made on TD types (see Section 3.1.1). From the gathered results, TD types appear with different frequencies in SO questions (see Figure 2). Architecture, Code, and Design TD are the most recurrent TD types, while all other types are mentioned approximately at least half less frequently. Overall, we expect the observed recurrence of TD types to reflect to a large extent the frequency with which practitioners face the TD types in practice. An observation nevertheless needs to be made regarding the high occurrence of Architecture TD, which we conjecture is in part influenced by the keyword-based strategy used to automatically identify TDQs (see Section 2.3.1). In fact, directly referencing the term "technical debt" in a TDQs might require a level of abstraction and reflection on programming/design processes which is more common at the architectural level, leading potentially to a higher appearance of Architecture TD in our TDQ corpus.

By considering systematic literature review of Li *et al.* [37], we note remarkable differences in the recurrence of TD types reported in such study and our results. For example, Requirements TD is the fourth most recurring TD type in our study, while it is among the least mentioned ones in the literature. We reason that such discrepancy highlights potential differences between academic research and industrial practice, the latter of which might be more closely represented in our study. This conjecture is further corroborated by the low occurrence in our TDQ corpus of the TD sub-categories elicited by Li *et al.* [37] (see also Section 2.3.4), indicating that different topics are more prominent in the literature and SO.

**Key finding 1:** Architecture, Code, and Design TD are the most referenced TD types on Stack Overflow. Other TD types are also referenced, but at least half less frequently. A notable difference in TD type recurrence with respect to the academic body of literature is observed.

Regarding the length of the TDQs (see Section 3.1.2), most TD types are characterized by comparable question length (see Table 1). Only Design and Test TD present slighter longer median questions.

Upon inspection of the Design TDQs, we conjecture that the slightly longer Design TD question length might be due to the characteristic need of presenting both higher level concepts, typical of the architectural level, and lower level ones, typical of the code level one. Similarly, regarding Test TD, we attribute the longer length to the recurrent need in Test TD questions of including both details regarding the test environment considered and code snippets used to exemplify the questions. By inspecting the Documentation TD questions instead, we observe that most questions are of general, opinion-based, and open-ended nature (e.g. "*What are the crucial key items in recording technical debt?*"[15]). Therefore, such questions do not require a high number of words to present specifics, details, or code snippets, leading to shorter questions. By considering the related literature, we note that all TD types showcase a medium question length (between 90 and 200 words), reflecting the common length of SO questions [14].

**Key finding 2:** All TD types reflect the common length of Stack Overflow questions. Design and Test TD are characterized by slightly longer questions w.r.t. the other types, while documentation TD by slightly shorter ones.

From the recorded levels of perceived urgency (see Section 3.1.3), the vast majority of TDQs display some degree of urgency, mostly being of very mild nature. The frequency of higher urgency questions steadily declines as urgency rises (see also Figure 3.1.3). On one hand, the observed distribution might partially reflect the commonly experienced urgency of TD items in practice. On the other hand, the distribution might be partially influenced by the very nature of the SO platform. Most questions display a mild degree of urgency, which justifies the effort spent in posting the questions on SO. Higher urgency questions instead are less frequent, potentially due to the asynchronous nature of SO, the slightly unpredictable quality of the answers, and difficulties related to comprehensively report complex questions in a time-efficient manner. All these aspects may lead higher urgency TD questions to be discussed in other venues (e.g, face to face meetings with other software developer colleagues), and not on SO.

Regarding the relative urgency across TD types (see Figure 5), Test TD presents most frequently high urgency questions, followed by Code and Versioning TD. By inspecting the high urgency questions of these types, we note that often the questions regard TD items that are currently hindering, to a large extent, normal development activities. We infer that such types of TD items more frequently occur in Test, Code and Versioning TD than for other TD types (e.g., Architecture or Requirements TD), which would justify the higher urgency of Test, Code and Versioning TD questions.

**Key finding 3:** Most TD questions display some degree of urgency. Frequency of higher urgency questions steadily declines as urgency rises. Test TD questions display the highest relative urgency, followed by Code and Versioning TD.

By considering the sentiment expressed in the questions (see Section 3.1.4), we note that most questions display primarily a neutral sentiment (see also Figure 6). While TD concepts might be intuitively associated to a potentially higher negative sentiment, given the technical nature of the Q&A platform considered, we deem this finding unsurprising. More interestingly, by considering the compound sentiment expressed in the questions (Figure 6), we observe that questions often tend to express an overall more positive than negative sentiment. Upon manual inspection of the questions with a higher compound sentiment, we conclude that this finding is not due to the sentiment expressed while discussing TD *per se*. In fact, the positive sentiment is often expressed in portions of the question body, commonly located towards the end of the questions, that users utilize to express their gratitude for the answers they received or expect to receive in the future. Therefore, we conjecture that the overall sentiment expressed in TD questions is generally of neutral nature, with positive sentiments driven primarily by the politeness utilized by users to pose questions.

**Key finding 4:** The sentiment expressed in TD questions is primarily of neutral nature, with slightly positive sentiments driven primarily by the politeness used to pose the questions.

The last characteristic we consider to answer $RQ_1$ are the salient themes considered in our TDQ corpus (see Section 3.1.5). From the collected results we can observe that the 29 recurrent themes (see also Figure 9) are characterized by different levels of granularity, ranging from topics of general nature, *e.g., TD resolution*, and *TD management*, to lower-level themes, *e.g., concurrency* and *warnings*.

*TD resolution* and *TD management* are themes mentioned across all TD types, and are the most recurring themes overall. Given the high level of abstraction such themes entail, we deem their high recurrence, and mapping to all TD types, as rather unsurprising. Instead, the high occurrence of the *software patterns*, *suboptimal design*, and *databases* themes results to be primarily driven by questions regarding Architecture TD, which is the most frequently appearing TD type in our TDQs corpus. By comparing our findings with the related literature which also utilizes SO as data source (see also Section 6), we note that *databases* are also among the most recurrent indicators used in TD identification processes [27], further supporting the high occurrence in our data of this rather specific theme.

Regarding the distribution of themes across TD types, we deem that the distribution reflects what could be intuitively expected. For example, *dependencies* among software components are mostly referenced at a high and medium levels of abstraction, *i.e.,* in questions related to Architecture and Design TD. In contrast, *coding standards* appear only at a medium or low levels of abstraction, *i.e.,* when discussing Design and Code TD. Similarly, the theme *frameworks* is mostly associated to the TD types where intuitively software frameworks might be discussed the most, namely Architecture and Infrastructure TD.

While most themes are supported, to various extents, by questions belonging to different TD types, we also notice that some themes are almost if not exclusively characteristic of specific TD types. For example, *branching hell*, *upgrading issues*, and TD due to utilizing *multiple versions*, are all themes exclusive to Versioning TD. Similarly, Test TD distinguishes itself from the other TD types by being the only one presenting the themes of *code coverage*, *testing complexity*, and most occurrences of *mocking*. By adopting a more encompassing search strategy (*e.g.,* based on our $RQ_2$ findings), it might be possible to further characterize TD types *via* the analysis of SO, *e.g.,* by identifying the TD items characteristic of each TD type.

As a separate observation, *software development methodologies*, which might be more discussed on Q&A sites dedicated to the

[15] https://stackoverflow.com/questions/3902389/what-are-the-crucial-key-items-in-recording-technical-debt. Accessed 26th April 2022.

software development life cycle (*e.g.,* the Software Engineering Stack Exchange[16]) than coding Q&A sites such as SO, appears in our results among the 10 most frequent themes. From a further inspection of the TDQs related to this theme, we note that most of these questions regard the integration of TD management practices into software development methodologies, with test driven development being often mentioned.

Finally, among TDQs themes, we also observed the presence of *TD communication, i.e.,* questions regarding the communication of concepts related to TD. To the best of our knowledge, this topic is only marginally investigated in academic literature (*e.g.,* in a theory on architecture debt Verdecchia *et al.* mention "communication" as a category [59]). Nevertheless, by following a reasoning similar to our discussion on *software development methodologies*, the presence of such topic also on SO, rather than more fitting platforms, could indicate that TD communication is a thematic which is more widespread in practice than what the related academic literature could suggest.

**Key finding 5:** 29 salient themes emerge from the questions. *TD resolution* and *TD management* are the most recurring themes, potentially due to their encompassing nature. Most themes are distributed across TD types, while a minor portion are characteristic to specific TD types.

Regarding $RQ_2$, although the manual analysis of SO posts is a useful technique for our study, automated approaches are needed for the analysis of TD in SO questions. This need is motivated by the large number of questions posted daily on SO and the high presence of non-technical debt related questions (see Section 2.3.2). In this respect, results answering $RQ_2$ are encouraging. TDQs can be filtered with an $F_1$ score of 0.83. Similarly, the binary categorization of the urgency of the TDQs (urgent, non-urgent) has an $F_1$ score of 0.84. Our results for the multi-class models are less encouraging. The categorization into the nine different TD types have an average $F_1$ score of 0.49, whereas the categorization of urgency into a six-level scale have an average $F_1$ score of 0.63. We believe that a larger labelled dataset could allow for the training of multi-class models with better accuracy. It is important to note that we avoid the over-fitting of the models by removing all technical-debt related keywords that are used for the collection of the original training and test set.

**Key finding 6:** Predictive models are able to detect and classify with high precision and recall TD questions and their binary urgency, but not TD types and six-level urgency.

## 5 THREATS TO VALIDITY

Despite our best efforts, the results presented in this study may be affected by validity threats. By following the classification of Runeson *et al.* [50], we consider four aspects to discuss the potential threats of this study and the related mitigation strategies adopted.

### 5.1 Construct Validity

Regarding the extent to which our operational measures are appropriate to answer our RQs. To answer $RQ_1$, five qualitative and quantitative TDQs characteristics of different nature are considered (see Section 2.3.3 and Section 2.3.4). The selection of characteristics was guided by the aspects of TDQ we deemed most relevant in the context of TD. We do not expect such selection to majorly hinder our answer to $RQ_1$. A related threat lies in the selection of the TD types used for the provisional coding phase (see Section 2.3.3). To mitigate potential threats, we use the well-established and widely-adopted taxonomy of TD types presented by Li *et al.* [37] for the provisional coding. A similar threat regards the adoption of a six-level urgency scale to rank the perceived urgency of TDQs. From a preliminary investigation, the six-level granularity proved to be intuitive yet satisfactorily expressive, and we not deem that adopting such scale considerably influenced our answer to $RQ_1$. Regarding $RQ_2$, we used *de facto* standard metrics to evaluate our classifiers (namely precision, recall, and $F_1$ score), which should nullify potential construct threats in answering $RQ_2$.

### 5.2 Internal Validity

Regarding the extent to which the observed results are due to the "treatment" and not to other factors. A threat of this category lies in the design of the automated query used to identify TDQs in Phase 1 (see Section 2.3.1). To mitigate potential threats related to the query design, we ensure to (i) include various synonyms of TD, (ii) consider not only the body of questions, but also their title, tags, and answers, and (iii) manually scrutinize all automated query results (see Section 2.3.2). Another threat to internal validity in our study is due to the manual coding steps entailed by Phase 2, Phase 3, and Phase 4a. To mitigate this threat we (i) use a coding guide, (ii) discuss coding discrepancies among researchers till a consensus is reached, (iii) assess inter-rater reliability of Phases 2-3, and (iv) conduct an examination of the coded data produced in Phase 4a.

### 5.3 External Validity

Regarding the extent to which our results are generalizable. In order to mitigate potential threats to external validity, the automated query used in Phase 1 was purposely designed to be as encompassing as possible, while leading to a preliminary question set which is with high probability relevant for our study. Nevertheless, our results may not be representative of the entirety of questions related to TD present on SO. In fact, our study does not include potential questions related to TD which do not contain the keyword "technical debt" (or variations thereof, see Section 2.3.1) in their title, body, tags, or answers. This threat may affect our results, and can be mitigated in future research by adopting other search strategies, *e.g.,* the one modeled in this study to answer $RQ_2$.

### 5.4 Reliability

Regarding the extent to which our observations can be reproduced by other researchers. To ensure reliability of our results, we make all data used in this paper, scripts, and relative settings, available in the replication package of this study (see Section 1).

## 6 RELATED WORK

In this section we present and discuss the literature that, to the best of our knowledge, is related the closest to this study. The work which resembles our study most, in terms of topic and used research

[16]https://softwareengineering.stackexchange.com. Accessed 27th April 2022.

process, is a study by Gama *et al.* [27]. In such study, Gama *et al.* manually analyze a curated sample of 140 Stack Overflow discussions to study how developers identify TD. As most prominent difference with such work, our study aims at providing a general overview of TD questions on SO, by including all identified questions concerning TD, regardless of the specific aspect of TD or TD process considered (*e.g.,* TD management [9, 29, 36] and TD prioritization [3, 20, 33]). In contrast, Gama *et al.* focus exclusively on one specific process related to TD, namely TD identification [5, 26, 60, 62], to study how TD items are commonly identified by developers [27]. As additional differences, our study considers as aspects to characterize TDQs the TD types taken from the taxonomy of Li *et al.* [37] (see Section 5.1 for further rationale on such selection), question length, perceived urgency, sentiment, and considered themes. Differently, while also considering TD types (albeit by using the taxonomy of Rios *et al.* [48]), the other aspects studied by Gama *et al.* differ, and focus specifically on TD identification aspects (namely low- and high-level TD indicators, and their relations to TD types). Finally, our study focuses also on the automatic identification of TDQs ($RQ_2$), while this aspect is not considered in the work of Gama *et al.*

Digkas *et al.* [22] study the relation between reusing code from SO and TD. While both our study and the one of Digkas *et al.* focus on TD, and consider SO as a data source, the two studies differ drastically. Specifically, we focus on the characteristics of TDQs (see Section 3.1) and their automatic classification (see Section 3.2), while Digkas *et al.* investigate, *via* source code static analysis, the technical debt incurred by reusing code snippets posted on the platform.

Another study related to TD and SO is the work by Perez *et al.* [43], where the authors propose a model-driven approach to manage the architectural technical debt life cycle. Rather than focusing on SO questions, in such study, a dataset of SO posts [13] is used to assess the sentiment expressed while documenting architectural design decisions.

With TDMentions [24], Ericsson and Wingkvist make available a dataset of TD mentions on various platforms, including the Stack Exchange network under which SO is provided. We opted to query directly the Stack Overflow Data Dump, rather than using TDMentions, in order to have full control over the query to be executed, tailor the query according to our focus, and ensure that the query satisfied our requirements (*e.g.,* in terms of keywords to be used and search fields to be considered, see Section 2.3.1).

Marginally related to our work, similar to the method used to answer $RQ_1$, numerous studies utilized SO to acquire knowledge on different topics related to software engineering, such as code smells [53–55], non-functional requirements [2, 11, 63], refactoring [4, 44, 45], API usage [1, 39, 42], and coding practices [23, 46, 57].

In a similar vein, some studies investigated the automatic identification of questions regarding various topics related to software engineering [1, 10, 11, 52, 56]. Such studies showcase a similar accuracy to the we report for $RQ_2$, while in some cases being able to detect questions at a finer level of granularity, potentially due to the usage of larger question sets.

Finally, we acknowledge two closely related studies, published in Portuguese, which focus on TD on SO. To avoid potential misinterpretations of the full-text, we refrain from carrying out an in-depth comparison, and base our discussion on the abstracts of such studies, which are provided in English. To the best of our understanding, in a first study Costa *et al.* [19] build upon the work of Gama *et al.* [27], to investigate the relation between technical debt indicators identified in the work of Gama *et al.* [27] and the quality attributes defined in the ISO/IEC 25010 standard [31]. The main differences with our study and the one by Costa *et al.* [19] are the same as those readily discussed for the work of Gama *et al.* [27], with the additional difference of quality attributes, which are not considered in our study. In a second study, Gama *et al.* [28] analyze 195 SO discussions on TD to study the recurrent TD types, activities, strategies, and tools used in TD management. In difference to such work, our study focuses on SO questions rather than discussions, considers a higher number of data points (see Section 2.3.2), and utilizes different aspects to characterize TD questions (see Section 3.1). In addition, our study focuses also on the automated identification of TDQs (see Section 3.2), an aspect not considered in the work of Gama *et al.* [28].

## 7 CONCLUSION AND FUTURE WORK

Our investigation provides empirical insights into the characteristics of technical debt questions posed on Stack Overflow. To achieve our goal, we study a curated set of 415 Stack Overflow questions regarding technical debt, by adopting a mix of qualitative and quantitative analyses. From the results clear patterns emerge regarding the frequency of technical debt types, the urgency expressed in the questions, their length, sentiments, and considered themes. In addition, we demonstrate how predictive models can successfully be used to automatically identify and classify technical debt questions.

For researchers, this study constitutes a first step, providing both quantitative and qualitative empirical insights, to understand how users reference and request support on technical debt online. Our results suggest that, by adopting a more comprehensive strategy to identify technical debt questions, further insights into the technical debt phenomenon can be gained by considering Stack Overflow. For example, technical debt items specific to each technical debt type, common resolutions, and management strategies could be determined. In addition, we support future research by making all our labeled data, and our predictive models for utilizing a more encompassing search strategy, available in our replication package.

For practitioners, our study provides concrete evidence of the most frequently referenced TD types, themes, and urgencies of TDQs. Such findings can corroborate or refine their understanding and use of Stack Overflow to ask TDQs, and access structured knowledge that could not be gained by considering their personal experience alone.

As future work, we plan to use the predictive models trained for this investigation to gain a more encompassing overview of technical debt from the viewpoint of Stack Overflow. In addition, we aim at including also other Q&A platforms, such as the Software Engineering Stack Exchange, to gain a more encompassing and less "code-centric" overview on how technical debt is discussed on online Q&A sites. Finally, regarding the automatic identification and classification of technical debt questions, it would be interesting to train models with a larger amount of data and evaluate models trained on TDQs focusing on specific software applications, technologies, and programming languages, *e.g.,* AI-based systems [12], Android apps [58], or scientific software [17]. These models might have a higher performance than the ones presented in this research, as they could learn about the specific software context considered in the TDQs.

## REFERENCES

[1] Md Ahasanuzzaman, Muhammad Asaduzzaman, Chanchal K Roy, and Kevin A Schneider. 2018. Classifying stack overflow posts on API issues. In *2018 IEEE 25th international conference on software analysis, evolution and reengineering (SANER)*. IEEE, 244–254.

[2] Arshad Ahmad, Chong Feng, Kan Li, Syed Mohammad Asim, and Tingting Sun. 2019. Toward empirically investigating non-functional requirements of iOS developers on stack overflow. *IEEE Access* 7 (2019), 61145–61169.

[3] Reem Alfayez, Wesam Alwehaibi, Robert Winn, Elaine Venson, and Barry Boehm. 2020. A systematic literature review of technical debt prioritization. In *Proceedings of the 3rd International Conference on Technical Debt*. 1–10.

[4] Eman Abdullah Alomar, Tianjia Wang, Vaibhavi Raut, Mohamed Wiem Mkaouer, Christian Newman, and Ali Ouni. 2022. Refactoring for reuse: an empirical study. *Innovations in Systems and Software Engineering* (2022), 1–31.

[5] Nicolli SR Alves, Thiago S Mendes, Manoel G de Mendonça, Rodrigo O Spínola, Forrest Shull, and Carolyn Seaman. 2016. Identification and management of technical debt: A systematic mapping study. *Information and Software Technology* 70 (2016), 100–121.

[6] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162). *Dagstuhl Reports* 6 (01 2016). https://doi.org/10.4230/DagRep.6.4.110

[7] Paris C Avgeriou, Davide Taibi, Apostolos Ampatzoglou, Francesca Arcelli Fontana, Terese Besker, Alexander Chatzigeorgiou, Valentina Lenarduzzi, Antonio Martini, Athanasia Moschou, Ilaria Pigazzini, et al. 2020. An overview and comparison of technical debt measurement tools. *IEEE Software* 38, 3 (2020), 61–71.

[8] Victor R. Basili, Gianluigi Caldiera, and Dieter Rombach. 1994. The Goal Question Metric Approach. In *Encyclopedia of Software Engineering*. Wiley, 528–532.

[9] Christoph Becker, Ruzanna Chitchyan, Stefanie Betz, and Curtis McCord. 2018. Trade-off decisions across time in technical debt management: a systematic literature review. In *Proceedings of the 2018 International Conference on Technical Debt*. 85–94.

[10] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2018. Automatically classifying posts into question categories on stack overflow. In *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 211–21110.

[11] Tingting Bi, Peng Liang, Antony Tang, and Xin Xia. 2021. Mining architecture tactics and quality attributes knowledge in Stack Overflow. *Journal of Systems and Software* 180 (2021), 111005.

[12] Justus Bogner, Roberto Verdecchia, and Ilias Gerostathopoulos. 2021. Characterizing technical debt and antipatterns in AI-based systems: A systematic mapping study. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 64–73.

[13] Fabio Calefato, Filippo Lanubile, Federico Maiorano, and Nicole Novielli. 2018. Sentiment polarity detection for software development. *Empirical Software Engineering* 23, 3 (2018), 1352–1382.

[14] Fabio Calefato, Filippo Lanubile, and Nicole Novielli. 2018. How to ask for technical help? Evidence-based guidelines for writing questions on Stack Overflow. *Information and Software Technology* 94 (2018), 186–207.

[15] Rafael Capilla, Tommi Mikkonen, Carlos Carrillo, Francesca Arcelli Fontana, Ilaria Pigazzini, and Valentina Lenarduzzi. 2021. Impact of Opportunistic Reuse Practices to Technical Debt. In *2021 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 16–25.

[16] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. 2002. SMOTE: synthetic minority over-sampling technique. *Journal of artificial intelligence research* 16 (2002), 321–357.

[17] Zadia Codabux, Melina Vidoni, and Fatemeh H Fard. 2021. Technical debt in the peer-review documentation of r packages: A rOpenSci case study. In *2021 IEEE/ACM 18th International Conference on Mining Software Repositories (MSR)*. IEEE, 195–206.

[18] Jacob Cohen. 1960. A coefficient of agreement for nominal scales. *Educational and psychological measurement* 20, 1 (1960), 37–46.

[19] Diego Costa, Mariela Cortés, and Eliakim Gama. 2021. On the relation between technical debt indicators and quality criteria in Stack Overflow discussions. In *Brazilian Symposium on Software Engineering*. 432–441.

[20] Rodrigo Rebouças de Almeida, Uirá Kulesza, Christoph Treude, Aliandro Higino Guedes Lima, et al. 2018. Aligning technical debt prioritization with business objectives: A multiple-case study. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 655–664.

[21] George Digkas, Alexander N Chatzigeorgiou, Apostolos Ampatzoglou, and Paris C Avgeriou. 2020. Can clean new code reduce technical debt density. *IEEE Transactions on Software Engineering* (2020).

[22] Georgios Digkas, Nikolaos Nikolaidis, Apostolos Ampatzoglou, and Alexander Chatzigeorgiou. 2019. Reusing code from stackoverflow: the effect on technical debt. In *2019 45th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 87–91.

[23] Maarten Duijn, Adam Kucera, and Alberto Bacchelli. 2015. Quality questions need quality code: Classifying code fragments on stack overflow. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 410–413.

[24] Morgan Ericsson and Anna Wingkvist. 2019. TDMentions: a dataset of technical debt mentions in online posts. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 123–124.

[25] Savio Freire, Nicolli Rios, Boris Perez, Camilo Castellanos, Dario Correal, Robert Ramač, Vladimir Mandić, Nebojša Taušan, Alexia Pacheco, Gustavo Lopez, et al. 2021. Pitfalls and Solutions for Technical Debt Management in Agile Software Projects. *IEEE Software* 38, 6 (2021), 42–49.

[26] Mário André de Freitas Farias, José Amâncio Santos, Marcos Kalinowski, Manoel Mendonça, and Rodrigo Oliveira Spínola. 2016. Investigating the identification of technical debt through code comment analysis. In *International Conference on Enterprise Information Systems*. Springer, 284–309.

[27] Eliakim Gama, Sávio Freire, Manoel Mendonça, Rodrigo O Spínola, Matheus Paixao, and Mariela I Cortés. 2020. Using Stack Overflow to Assess Technical Debt Identification on Software Projects. In *Proceedings of the 34th Brazilian Symposium on Software Engineering*. 730–739.

[28] Eliakim Gama, Matheus Paixao, Emmanuel Sávio Silva Freire, and Mariela Inés Cortés. 2019. Technical Debt's State of Practice on Stack Overflow: a Preliminary Study. In *Proceedings of the XVIII Brazilian Symposium on Software Quality*. 228–233.

[29] Yuepu Guo, Rodrigo Oliveira Spínola, and Carolyn Seaman. 2016. Exploring the costs of technical debt management–a case study. *Empirical Software Engineering* 21, 1 (2016), 159–182.

[30] Clayton Hutto and Eric Gilbert. 2014. Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media*, Vol. 8. 216–225.

[31] ISO/IEC 25010:2011. 2011. ISO/IEC 25010:2011, Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models.

[32] Clemente Izurieta, Antonio Vetrò, Nico Zazworka, Yuanfang Cai, Carolyn Seaman, and Forrest Shull. 2012. Organizing the technical debt landscape. In *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 23–26.

[33] Valentina Lenarduzzi, Terese Besker, Davide Taibi, Antonio Martini, and Francesca Arcelli Fontana. 2021. A systematic literature review on technical debt prioritization: Strategies, processes, factors, and tools. *Journal of Systems and Software* 171 (2021), 110827.

[34] Valentina Lenarduzzi, Francesco Lomio, Nyyti Saarimäki, and Davide Taibi. 2020. Does migrating a monolithic system to microservices decrease the technical debt? *Journal of Systems and Software* 169 (2020), 110710.

[35] Valentina Lenarduzzi, Nyyti Saarimaki, and Davide Taibi. 2019. On the diffuseness of code technical debt in java projects of the apache ecosystem. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 98–107.

[36] Jean-Louis Letouzey and Michel Ilkiewicz. 2012. Managing technical debt with the sqale method. *IEEE software* 29, 6 (2012), 44–51.

[37] Zengyang Li, Paris Avgeriou, and Peng Liang. 2015. A systematic mapping study on technical debt and its management. *Journal of Systems and Software* 101 (2015), 193–220.

[38] Bin Lin, Fiorella Zampetti, Gabriele Bavota, Massimiliano Di Penta, Michele Lanza, and Rocco Oliveto. 2018. Sentiment analysis for software engineering: How far can we go?. In *Proceedings of the 40th international conference on software engineering*. 94–104.

[39] Mario Linares-Vásquez, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, and Denys Poshyvanyk. 2014. How do api changes trigger stack overflow discussions? a study on the android sdk. In *proceedings of the 22nd International Conference on Program Comprehension*. 83–94.

[40] Walaa Medhat, Ahmed Hassan, and Hoda Korashy. 2014. Sentiment analysis algorithms and applications: A survey. *Ain Shams engineering journal* 5, 4 (2014), 1093–1113.

[41] Sebastian Ospina, Roberto Verdecchia, Ivano Malavolta, and Patricia Lago. 2021. ATDx: A tool for providing a data-driven overview of architectural technical debt in software-intensive systems. In *European Conference on Software Architecture*.

[42] Chris Parnin, Christoph Treude, Lars Grammel, and Margaret-Anne Storey. 2012. Crowd documentation: Exploring the coverage and the dynamics of API discussions on Stack Overflow. *Georgia Institute of Technology, Tech. Rep* 11 (2012).

[43] Boris Pérez, Darío Correal, and Hernán Astudillo. 2019. A proposed model-driven approach to manage architectural technical debt life cycle. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 73–77.

[44] Anthony Peruma, Steven Simmons, Eman Abdullah AlOmar, Christian D Newman, Mohamed Wiem Mkaouer, and Ali Ouni. 2022. How do i refactor this? An empirical study on refactoring trends and topics in Stack Overflow. *Empirical Software Engineering* 27, 1 (2022), 1–43.

[45] Gustavo H Pinto and Fernando Kamei. 2013. What programmers say about refactoring tools? an empirical investigation of stack overflow. In *Proceedings of the 2013 ACM workshop on Workshop on refactoring tools*. 33–36.

[46] Chaiyong Ragkhitwetsagul, Jens Krinke, Matheus Paixao, Giuseppe Bianco, and Rocco Oliveto. 2019. Toxic code snippets on stack overflow. *IEEE Transactions on Software Engineering* 47, 3 (2019), 560–581.

[47] Anand Rajaraman and Jeffrey David Ullman. 2011. *Data Mining*. Cambridge University Press, 1–17. https://doi.org/10.1017/CBO9781139058452.002

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

[49] Nicolli Rios, Savio Freire, Boris Perez, Camilo Castellanos, Dario Correal, Manoel Mendonca, Davide Falessi, Clemente Izurieta, Carolyn B Seaman, and Rodrigo Oliveira Spinola. 2021. On the relationship between technical debt management and process models. *IEEE Software* 38, 5 (2021), 56–64.

[50] Per Runeson and Martin Höst. 2009. Guidelines for conducting and reporting case study research in software engineering. *Empirical software engineering* 14, 2 (2009), 131–164.

[51] Johnny Saldaña. 2021. *The coding manual for qualitative researchers*. Sage, 291–298.

[52] Sergei Shcherban, Peng Liang, Amjed Tahir, and Xueying Li. 2020. Automatic identification of code smell discussions on stack overflow: A preliminary investigation. In *Proceedings of the 14th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM)*. 1–6.

[53] Amjed Tahir, Jens Dietrich, Steve Counsell, Sherlock Licorish, and Aiko Yamashita. 2020. A large scale study on how developers discuss code smells and anti-pattern in stack exchange sites. *Information and Software Technology* 125 (2020), 106333.

[54] Amjed Tahir, Aiko Yamashita, Sherlock Licorish, Jens Dietrich, and Steve Counsell. 2018. Can you tell me if it smells? a study on how developers discuss code smells and anti-patterns in stack overflow. In *Proceedings of the 22nd International Conference on Evaluation and Assessment in Software Engineering 2018*. 68–78.

[55] Fangchao Tian, Peng Liang, and Muhammad Ali Babar. 2019. How developers discuss architecture smells? an exploratory study on stack overflow. In *2019 IEEE international conference on software architecture (ICSA)*. IEEE, 91–100.

[56] Fangchao Tian, Fan Lu, Peng Liang, and Muhammad Ali Babar. 2020. Automatic Identification of Architecture Smell Discussions from Stack Overflow.. In *SEKE*. 451–456.

[57] Christoph Treude and Martin P Robillard. 2017. Understanding stack overflow code fragments. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 509–513.

[58] Roberto Verdecchia. 2018. Identifying architectural technical debt in android applications through automated compliance checking. In *2018 IEEE/ACM 5th International Conference on Mobile Software Engineering and Systems (MOBILESoft)*. IEEE, 35–36.

[59] Roberto Verdecchia, Philippe Kruchten, Patricia Lago, and Ivano Malavolta. 2021. Building and evaluating a theory of architectural technical debt in software-intensive systems. *Journal of Systems and Software* 176 (2021), 110925.

[60] Roberto Verdecchia, Ivano Malavolta, and Patricia Lago. 2018. Architectural technical debt identification: The research landscape. In *2018 IEEE/ACM International Conference on Technical Debt (TechDebt)*. IEEE, 11–20.

[61] Roberto Verdecchia, Ivano Malavolta, Patricia Lago, and Ipek Ozkaya. 2022. Empirical evaluation of an architectural technical debt index in the context of the Apache and ONAP ecosystems. *PeerJ Computer Science* 8 (2022), e833.

[62] Nico Zazworka, Antonio Vetro, Clemente Izurieta, Sunny Wong, Yuanfang Cai, Carolyn Seaman, and Forrest Shull. 2014. Comparing four approaches for technical debt identification. *Software Quality Journal* 22, 3 (2014), 403–426.

[63] Jie Zou, Ling Xu, Weikang Guo, Meng Yan, Dan Yang, and Xiaohong Zhang. 2015. Which non-functional requirements do developers focus on? an empirical study on stack overflow using topic analysis. In *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*. IEEE, 446–449.