A Systematic Mapping Study Exploring Quantification Approaches to Code, Design, and Architecture Technical Debt

JUDITH PERERA*, School of Computer Science, The University of Auckland, New Zealand
EWAN TEMPERO, School of Computer Science, The University of Auckland, New Zealand
YU-CHENG TU, School of Computer Science, The University of Auckland, New Zealand
KELLY BLINCOE, Department of Electrical, Computer and Software Engineering, The University of Auckland, New Zealand

To effectively manage Technical Debt (TD), we need reliable means to quantify it. We conducted a Systematic Mapping Study (SMS) where we identified 39 quantification approaches for Code, Design, and Architecture TD. We analyzed concepts and metrics discussed in these quantification approaches by classifying the quantification approaches based on a set of abstract TD Quantification (TDQ) concepts and their high-level themes, process/time, cost, benefit, probability, and priority, which we developed during our SMS. This helped identify gaps in the literature and to propose future research directions. Among the abstract TDQ concepts discussed in the different quantification approaches, TD item, TD remediation cost, TD interest, and Benefit of remediating TD were the most frequently discussed concepts. They were also supported by some form of measurement. However, some TDQ concepts were poorly examined, for example, the benefit of taking TD. It was evident that cost concepts were more frequently quantified among the approaches, while benefit concepts were not. Most of the approaches focused on remediating TD in retrospect rather than quantifying TD to strategically use it during software development. This raises the question of whether existing approaches reliably quantify TD and suggests the need to further explore TD quantification.

CCS Concepts: • **Software and its engineering** → *Software design tradeoffs.*

Additional Key Words and Phrases: technical debt management, technical debt quantification, technical debt measurement, software quality

ACM Reference Format:
Judith Perera, Ewan Tempero, Yu-Cheng Tu, and Kelly Blincoe. 2024. A Systematic Mapping Study Exploring Quantification Approaches to Code, Design, and Architecture Technical Debt. *ACM Trans. Softw. Eng. Methodol.* 1, 1, Article 1 (July 2024), 49 pages. https://doi.org/10.1145/3675393

1 INTRODUCTION

Technical Debt (TD) captures the consequences of making sub-optimal decisions during software development [19]. While taking on TD can be advantageous in the short-term, for example, to deliver a product on time to market, it can lead to long-term degradation of the software product's quality. This can make it challenging to add new features and require additional work to implement them when there is TD [10, 11]. Therefore, TD must be prudently managed during software development

Authors' addresses: Judith Perera, jper@aucklanduni.ac.nz, School of Computer Science, The University of Auckland, Auckland, New Zealand, 1010; Ewan Tempero, e.tempero@auckland.ac.nz, School of Computer Science, The University of Auckland, Auckland, New Zealand, 1010; Yu-Cheng Tu, yu-cheng.tu@auckland.ac.nz, School of Computer Science, The University of Auckland, Auckland, New Zealand, 1010; Kelly Blincoe, k.blincoe@auckland.ac.nz, Department of Electrical, Computer and Software Engineering, The University of Auckland, Auckland, New Zealand, 1010.

1049-331X/2024/7-ART1 $15.00
https://doi.org/10.1145/3675393

[28, 60]. To effectively manage TD, it is essential to quantify it since quantifying TD can help make informed decisions regarding TD management (TDM) — *"In order to manage technical debt, a way to quantify the concept is needed"* [30]. We are investigating how TD can be quantified to better support TDM decision-making. In this paper, we explore existing proposals made in the literature to quantify Code, Design, and Architecture TD, through a *Systematic Mapping Study (SMS)*.

The SMS goal was to identify approaches attempting to quantify TD for the TD types Code, Design, and Architecture, and to learn how they discuss the quantification of TD and if they provide some form of measurement. We proposed a definition for a quantification approach to identify them from the various approaches attempting to perform different TD management activities such as identification, prioritization, monitoring, and repayment [39]. The definition is, *"An approach that discusses concepts and metrics that could support TDM decision-making"*. The Research Question for the SMS was **"RQ1: What approaches to quantifying Code, Design, and Architecture TD have been proposed in the research literature and how do they quantify TD?"**.

We identified 39 quantification approaches (QAs) and classified them based on a classification scheme that comprises a set of abstract TD Quantification (TDQ) concepts and their high-level themes, *process/time, cost, benefit, probability,* and *priority*, which we developed during the SMS. Classifying the QAs based on this classification scheme helped identify gaps in the existing QAs in terms of the abstract TDQ concepts they quantified and whether they proposed metrics that supported the quantification of those concepts.

According to our findings, most QAs commonly discuss the abstract TDQ concepts, *TD Item, TD Remediation Cost, TD Interest, and Benefit of remediating TD*, regardless of the TD type they quantify. Concepts, *TD Remediation Cost* and *TD Interest* are well supported by metrics. However, our findings suggested that most QAs focused on quantifying cost concepts, and the benefit concepts were under-explored. Furthermore, most approaches discussed quantifying TD retrospectively, focusing on the cost to remediate TD when TD exists in the system. The possibility of using TD strategically during software development was not adequately explored.

Given the map of classified QAs, practitioners can identify potential existing quantification approaches that fit their purpose, depending on what concepts they want to quantify to make decisions regarding TDM. However, whether the existing QAs are reliable means for a complete assessment of making TDM decisions is still an open question that needs further investigation — the consolidated set of abstract TDQ concepts we developed in this study can serve as a basis for further investigating this.

This paper reports on the methodology and results of the mapping study and discusses the implications of the findings, suggesting potential future work for researchers and implications for practitioners. The main contributions of this paper can be summarized as follows;

- **A Systematic Mapping Study (SMS)** — Methodology and results are a typical contribution of a SMS. We report the results of identifying and classifying quantification approaches for Code, Design, and Architecture TD. Additionally, we report the detailed methodology followed for the SMS to improve the repeatability of the study.
- **A set of abstract TDQ concepts sufficient to capture TD quantification conceptually** — A set of abstract TDQ concepts was developed during our study. They were further categorized into their high-level themes: process/time, cost, benefit, probability, and priority. The set of abstract TDQ concepts sufficiently captures what is required to understand the quantification of TD and can be considered a unified representation of the quantification approaches found in our SMS. Hence, they can serve as a reference to develop new quantification approaches.
- **A concept-based classification scheme for the classification of TD Quantification Approaches (QAs) for Code, Design, and Architecture TD** — The classification scheme

is based on the set of abstract TDQ concepts and their high-level themes that we developed during the SMS. The classification scheme was used to classify existing quantification approaches in our study. However, it can be reused to classify new quantification approaches. Hence, the classification scheme becomes a contribution on its own.
- **Future research directions for researchers and implications for practitioners** — We suggest future research directions for the research community derived from the implications of the findings of this study. Furthermore, we discuss potential implications for practitioners.

The rest of the paper is structured as follows. SMS methodology is reported in Section 2 while SMS results are discussed in Section 3. In Section 4, we discuss the implications of findings and future research directions and provide implications for practitioners. Section 5 discusses threats to the validity of our study. Section 6 discusses related work. The paper is concluded in Section 7.

2 METHODOLOGY

We followed recommendations given by Kitchenham et al. [34] and Petersen et al. [49, 50] to conduct our Systematic Mapping Study (SMS). The primary goal of a SMS is to structure a research area [50]. Our goal was to identify and classify quantification approaches proposed in the research literature for quantifying Code, Design, and Architecture TD, to learn what they discuss in terms of quantifying TD and if they provide some form of measurement for that. The definition we propose for a 'quantification approach' is: *"An approach that discusses concepts and metrics that could support TDM decision-making"*. Figure 1 shows the overview of the methodology and the number of papers retrieved at each stage of our mapping study. The following subsections describe the methodology. Further details can be found in our Replication Package[1].

2.1 Research Questions

The research question for the SMS was defined following the structure of the example of a typical SMS research question described in Petersen et al. [50];

- **RQ1: What approaches to quantifying Code, Design, and Architecture TD have been proposed in the research literature and how do they quantify TD?**
  - **RQ1.1: What TD Quantification (TDQ) Concepts do they discuss?**
  - **RQ1.2: Do they provide some form of measurement to the TDQ concepts discussed?**

2.2 Search

Articles accepted in the TechDebt conference in 2018 and 2019 (obtained from their website)[1] were used as a reference set to build our search string. Search terms were extracted from the title, abstract, keywords, and the full text of the articles in the reference set. Afterward, the search terms were expanded with synonyms. We tested the search phrase in SCOPUS, which retrieved all the articles in our reference set in the search results along with other articles.

The search string contained the following terms and synonyms in its final version (See Listing 1): *Technical Debt, quantify, measure, forecast, predict, assess, estimate, calculate, amount, value, impact, principal, interest, metric, time, cost*. We used the term 'Technical Debt' along with the rest of the keywords to avoid articles not focusing on TD (e.g., articles on software quality or architecture but not TD). We used the asterisk character *(*)* to capture possible variations of the keywords, such as

[1]Replication Package: https://doi.org/10.5281/zenodo.10617774
[1]Papers accepted in Tech Debt 2018, 2019: https://2018.techdebtconf.org/#event-overview, https://2019.techdebtconf.org/#Accepted-Papers

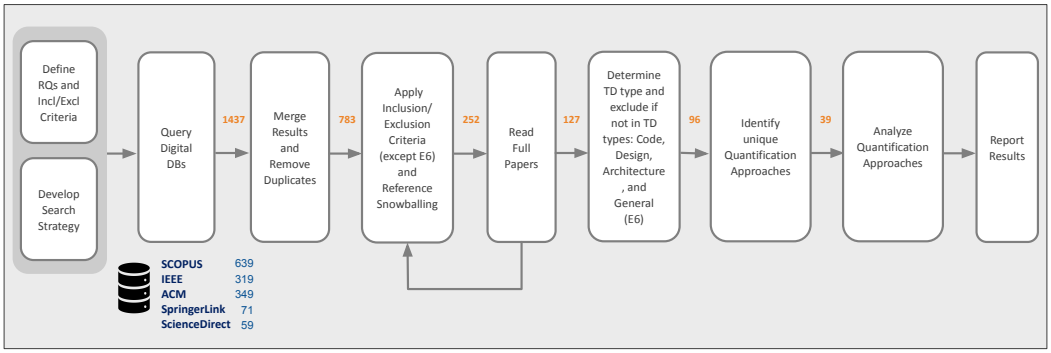

Fig. 1. SMS Methodology Overview and Results retrieved at each Stage

plurals and verb conjugations. We applied the query to the article's title, abstract, and keywords to increase the probability of finding all relevant articles. Also, we did not limit our search query to search for articles that mentioned only the TD types Code, Design, and Architecture, as we were aware that some studies did not mention a TD type and discussed TD in general but could be categorized as Code, Design or Architecture TD. Hence, we performed the categorization of articles into the TD types Code, Design, and Architecture while reading the full text of the articles (See Figure 1).

```
TITLE-ABS-KEY ( "Technical Debt"
        AND ( quantif*
                OR measur*
                OR forecast*
                OR predict*
                OR assess*
                OR estimat*
                OR calculat*
                OR impact*
                OR amount
                OR valu*
                OR principal
                OR interest*
                OR metric*
                OR time
                OR cost* ) )
        AND ( LIMIT-TO ( DOCTYPE , "cp" )
                OR LIMIT-TO ( DOCTYPE , "ar" ) )
        AND ( LIMIT-TO ( LANGUAGE , "English" ) )
```

Listing 1. Final Search String for SCOPUS

We tailored the final search string for the rest of the digital databases according to the functionality and usability of their interfaces. Digital databases IEEEXplore, ACM, and Science Direct,

were recommended by Brereton et al. [14], while SCOPUS was recommended by Cavacini [15]. Additionally, we queried SpringerLink. The search phase was concluded in September 2022.

All queries were performed by the first author. The search string was discussed and agreed with the rest of the authors prior to performing the search. Results from the multiple digital databases were merged, and duplicates were removed prior to study selection.

2.3 Study Selection

Article screening and selection was performed by the first author by applying the inclusion/exclusion criteria listed in Table 1. The inclusion and exclusion criteria were defined based on the reference set of articles [1] and then discussed and agreed upon with the other authors prior to performing the screening of the articles. When doubt was encountered during the screening of articles, they were discussed and resolved during consensus meetings with the other authors. The *adaptive reading depth approach* was followed for screening articles as suggested in Petersen et al. [49] starting from the title and then continuing through the abstract, conclusion, and at last, reading the full text.

2.3.1 Inclusion and Exclusion. Articles that discussed quantifying TD and articles that evaluated quantifying TD were included. Articles that described quantifiable characteristics of TD (e.g., principal, interest, interest probability) or units of measurement (in terms of; time, cost, or effort) were included. Since we were interested in the applicability of software metrics in the measurement of TD, we included articles discussing software metrics concerning TDM.

We did not consider secondary or tertiary studies as they would count the primary studies multiple times. We included only peer-reviewed articles as they are considered of high quality. We ruled out articles not written in English since we were not confident in other languages. Research articles not directly related to quantifying TD, i.e., articles related to measuring software quality but not related to TD, articles describing other TDM activities such as identification or prioritization but not quantification of TD, were ruled out. Articles that we could not access the full text were also ruled out. However, exclusion criterion E6 could be applied only after reading the full text since we had to determine the TD type for most of the articles that did not explicitly mention the type of TD, by reading the complete article.

Table 1. Inclusion and Exclusion Criteria

Inclusion Criteria	I1 Discusses quantifying TD I2 Discusses quantifiable characteristics of TD I3 Discusses units of measurement for TD I4 Discusses software metrics in relation to TDM I5 Evaluates quantifying TD
Exclusion Criteria	E1 Not a primary study E2 Not peer reviewed E3 Not written in English E4 Research is not directly related to Quantifying TD E5 Full text is inaccessible E6 Discusses a TD type that is not related to Code, Design, and Architecture E7 Discusses other TDM activities but not TD quantification

[1] Papers accepted in Tech Debt 2018, 2019: https://2018.techdebtconf.org/#event-overview, https://2019.techdebtconf.org/#Accepted-Papers

2.3.2 Reference Snowballing. Backward reference snowballing [64] was conducted on the included articles as an additional step to avoid the possibility of missing articles through only conducting a database search [50]. The results of snowballing were run through the Inclusion and Exclusion criteria before including them in the final set of articles. The process was iterated over these newly found articles until there were no more candidates for inclusion.

2.3.3 Identification of unique Quantification Approaches (QAs). We identified 96 articles discussing TD quantification in general. However, while examining the articles, we observed that some articles discussed TD identification or prioritization but did not necessarily discuss a form of measurement or quantification. Therefore, we applied the definition for a quantification approach: *"An approach that discusses concepts and metrics that could support TDM decision-making"*, to select the articles that could be categorized as quantification approaches based on our definition. In some cases, the same quantification approach was discussed in multiple articles; for example, while one article was introducing the approach, another evaluated it. We categorized those articles according to distinct quantification approaches. As an example, from the primary studies P105 (Notation — P[n]) and P108, we selected P105 as the unique quantification approach since P108 was based on P105 and was evaluating the use of the method introduced in P105.

As a result of performing both actions (excluding by definition and including only the most relevant article from a group of articles discussing the same approach), we identified 39 unique approaches to TD Quantification. See Figure 1 for the overview of the methodology and Table 5 in the appendix for the list of identified TD quantification approaches and their citations. The list of Primary Studies can be found in our Replication Package[1].

2.4 Data Extraction and Analysis

2.4.1 Classification Scheme. A typical classification scheme for a SMS could either be utilizing an existing scheme, for example, reporting data that is not topic specific, or a classification scheme emerging from the papers found in the mappings study, reporting on data that is topic specific [50]. In our case, we did the latter.

We developed a new classification scheme identifying themes, i.e., Abstract TDQ Concepts (See Figure 2) based on the initial codes, i.e., TDQ concepts extracted from the full text of the QAs with our Research Question in mind. We adopted, in part, the *keywording approach* recommended by Petersen et al. [50] and in part, the *Thematic Analysis* approach [18], a foundational method for qualitative data analysis originating from psychology. The keywording approach is a recursive process quite similar in nature to open coding in grounded theory [50]. Our coding process adopted the keywording approach but examined the full text of papers instead of abstracts. Our approach bears more similarity to *Thematic Analysis*, which identifies patterns (themes) within the data and minimally organizes and describes the dataset in rich detail [18].

In thematic analysis, a theme captures something important about the data in relation to the research question — our abstract TDQ concepts (themes) and their high-level categories (high-level themes) do this. See Figure 2 for the Classification Scheme resulting from the coding process (the coding process is further described below). We use it to classify and analyze QAs in Section 3.

The Coding Process. Our *Dataset* is the set of Quantification Approaches (QAs) resulting from the screening process of the SMS (See Figure 1). A *Data Item* is one QA, i.e., the paper that pertains to the QA. A *Data Extract* is a chunk of data that has been identified within and extracted from a data item, which could be either a sentence, a paragraph, or a set of paragraphs in the full text of the paper.

[1]Replication Package: https://doi.org/10.5281/zenodo.10617774

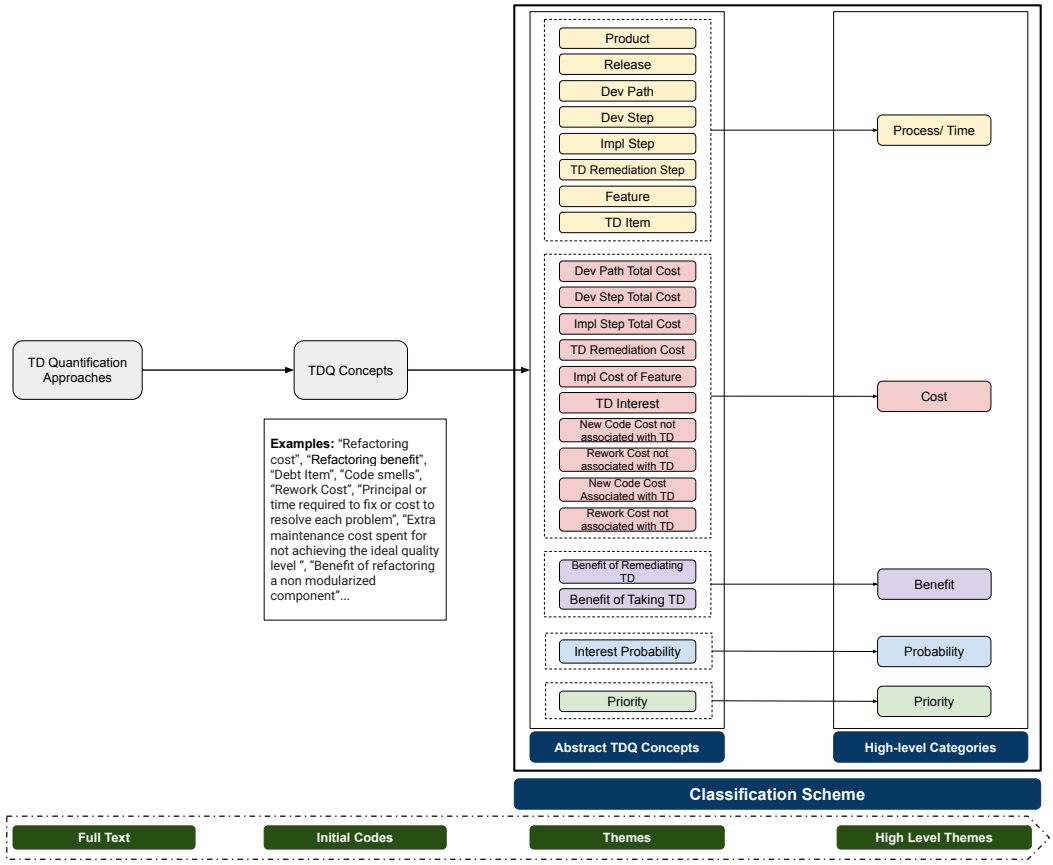

Fig. 2. Overview of the coding process and the resulting Classification Scheme — Abstract TDQ concepts are described in Table 2.

We read the article's title, abstract, keywords, RQs, introduction, and conclusion, and then read in detail the sections of the article where we saw relevant information and, lastly, the complete article to perform the data extraction and coding, following the adaptive depth reading approach recommended by Petersen et al. [50]. We identified and extracted various TDQ concepts as initial codes from the QAs. We mapped the TDQ concepts extracted from the QAs to a set of abstract TDQ concepts (i.e., themes) that we identified as recurring themes among the various TDQ concepts extracted from the QAs. *An abstract TDQ concept succinctly captures the notion described by multiple various concepts extracted from the QAs.* See Table 2 for their descriptions.

We traversed all QAs iteratively, extracting new concepts, abstracting them, and then re-traversing the QAs when a new abstract TDQ concept was introduced. Every time an existing abstract TDQ concept was not able to capture the concept extracted from a QA, a new abstract concept was introduced. After mapping concepts extracted from 33 out of 39 QAs, we did not have to introduce any more new abstract TDQ concepts. We could further categorize the abstract TDQ concepts into high-level categories (i.e., higher-level themes): process/time, cost, benefit, probability, and priority (See Figure 2).

The first author performed the coding process and developed the classification scheme. The other authors checked the initial codes and the mappings to abstract TDQ concepts (i.e., themes) and higher-level themes during iterative meetings. Table 2 describes the list of 22 abstract TDQ Concepts. An illustrative example of the coding is discussed with reference to Figure 3 below. The SMS classification scheme (See Figure 2) was discussed and agreed upon among all authors during the iterative consensus meetings prior to using it to classify and analyze the QAs.

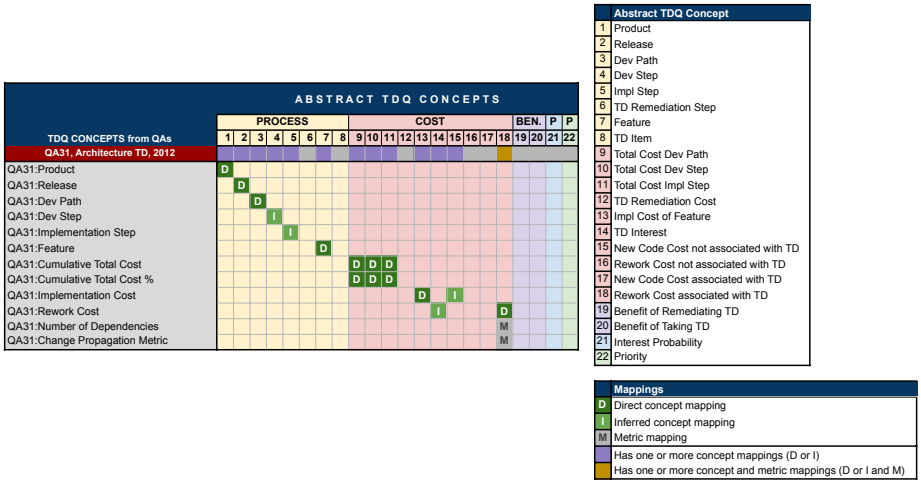

Fig. 3. Concept mapping for Nord et al. (QA31, Notation — QA[n])

We introduced two types of codes (i.e., mappings — we will use the term 'mapping' in the remainder of this paper) to map the initial codes extracted from the full texts of the QAs to the abstract TDQ concepts. The two types of codes were defined as 'concept' and 'metric,' as they allowed distinguishing between concepts and metrics identified from the full text of the QAs.

However, for the type 'concept,' not every concept extracted from the QA precisely mapped to the abstract TDQ concept we identified. Therefore, we introduced two sub-types of codes for the type 'concept,' Direct (D) and Inferred (I), to show how closely they relate to the identified abstract TDQ concept — *'the degree to which a given approach concept maps to an abstract TDQ concept'*.

The type 'metric' (M) was defined to indicate the mapping between *software metrics* described in the quantification approach and the abstract TDQ concepts if we could determine that the approach provided some form of measurement to quantify the abstract TDQ concepts.

- **'Direct' concept mapping — indicated by 'D' in Figure 3**: The approach concept corresponds exactly to the abstract TDQ concept, and it is straightforward to extract.
- **'Inferred' concept mapping — indicated by 'I'**: The approach concept relates to the abstract TDQ concept in some way but does not correspond exactly. i.e., it is inferred and not straightforward to extract.
- **'Metric' mapping — indicated by 'M'**: The approach provides some form of measurement that contributes to the quantification of the abstract TDQ concept.

Illustrative Example of the Mappings. Figure 3 illustrates the mapping of the concepts of an example quantification approach (QA) to the abstract TDQ concepts. In Figure 3, the approach concepts, i.e., concepts related to TD quantification extracted from the example approach Nord et al., QA31 (notation — QA[n]), are listed in the ***rows*** prefixed with *QA31*. Each approach concept

(prefixed with 'QA31:') is mapped individually to the abstract TDQ Concepts in the ***columns*** denoted 1-22. The legend can be found in the column on the right, next to the matrix. Each cell in the matrix that pertains to the intersection of an approach concept and an abstract TDQ concept is assigned a label for either a concept mapping 'D' or 'I' (according to whether the approach concept corresponds exactly to the abstract TDQ concept or if it was inferred that they relate) or, a metric mapping 'M' if the approach concept is a form of measurement for the abstract TDQ concept.

To describe this further, Nord et al. explicitly discuss 'Product'; therefore, the cell is labeled as 'D,' indicating the direct mapping to the abstract TDQ concept 'Product' (See Figure 3). In comparison, 'Rework Cost' is given mapping 'I' to the abstract TDQ concept 'Interest', indicating that we inferred this. However, 'Rework cost' is given mapping 'D' to the abstract TDQ concept 'Rework cost associated with TD' since it corresponds exactly to that abstract TDQ concept. 'Number of Dependencies' and 'Change propagation metric' are given mapping 'M' since they are metrics that contribute to the measurement of 'Rework Cost associated with TD'.

2.4.2 Classifying and Analyzing Quantification Approaches. We classified the quantification approaches (QAs) based on the concept-based classification scheme we developed based on the abstract TDQ concepts and their high-level categories process/time, cost, benefit, probability, and priority. This allowed for identifying gaps in the research literature as to which abstract TDQ concepts were discussed (or not) in the QAs based on the concept mappings and whether they were supported by any means of measurement based on the metric mapping. The classification is illustrated in Figures 8, 9, 10, 11, 12, and 13, and the analyzed results are discussed in Section 3.

Table 2. Abstract TDQ Concepts

Abstract TDQ Concept	Description	Category
Product	A software product developed by a software company	Process or time related
Release	A software product is delivered to the market through releases	Process or time related
Development Path (Dev Path)	The path developers follow to develop a software product	Process or time related
Development Step (Dev Step)	A step in the product development path where developers may work on implementation or debt remediation activities	Process or time related
Implementation Step (Impl Step)	A development step where feature implementation is performed	Process or time related
TD Remediation Step	A development step where developers remediate TD	Process or time related
Feature	A collection of user requirements delivered as a software functionality	Process or time related
TD Item	"A TD item is associated with one or more concrete, tangible artifacts of the software development process, primarily the code, but also to some extent the documentation, known defects, and tests associated with the system" [8]	Process or time related
Dev Path Total Cost	Total cost of a development path	Cost
Dev Step Total Cost	Total cost of a development step	Cost
Impl Step Total Cost	Total implementation cost	Cost

TD Remediation Cost	The cost to remediate TD during a remediation step	Cost
Impl Cost of Feature	The cost to implement a feature during an implementation step	Cost
TD Interest	The additional cost incurred due to the presence of technical debt [8]	Cost
New Code Cost not associated with TD	Cost of having to write new code but not as a consequence of TD	Cost
Rework Cost not associated with TD	Cost of having to do rework but not as a consequence of TD	Cost
New Code Cost associated with TD	Cost of having to write new code as a consequence of TD	Cost
Rework Cost associated with TD	Cost of having to do rework as a consequence of TD	Cost
Benefit of Remediating TD	Benefit gained by remediating TD	Benefit
Benefit of taking TD	Benefit gained by taking on TD as a strategy, e.g., as a strategy to gain competitive advantage by delivering a software product early to market	Benefit
Interest Probability	Whether or not implementation is affected by a TD item [16]	Probability
Priority	Priority in terms of severity or the impact a TD item may have on the code or the priority to remediate a TD Item	Priority

3 RESULTS: APPROACHES TO QUANTIFYING CODE, DESIGN, AND ARCHITECTURE TD (RQ1)

A Systematic Mapping Study (SMS) typically reports results in the order; first, the results of demographics that describe metadata of the studies discussed in the SMS, and then the findings obtained via the understanding of the studies. Following the same format, we report the results for demographics of primary studies and Quantification Approaches (QAs) found in our SMS in Sections 3.1, 3.2 and then the findings obtained for QAs in Sections 3.3 (RQ1, RQ1.1, RQ1.2), 3.4 (RQ1.1) and 3.5 (RQ1.2). Findings for the QAs are discussed based on the abstract TDQ concept-based classification we developed (See Figures 8, 9, 10, 11, 12, and 13 — they are described in Section 3.3).

3.1 Demographics of Primary Studies

3.1.1 Publication Year. We categorized primary studies based on their publication year (See Figure 4). Publications ranged from 2011 to 2022. The highest number of publications was found in 2018 — 16 primary studies in total, 6 of them discussing Architecture TD, 3 of them Code TD, 2 of them Design TD, and 5 of them belonged in the General category where they did not explicitly specify the type of TD, but we could infer them as related to Code, Design or Architecture TD. See Table 3 for the types of TD. The year 2011 is where the least number of studies were found (4 primary studies) after 2022 (3 primary studies), when the search phase of our mapping study was concluded. This can be because TD was an emerging research area at the time. In 2016 and 2018, there has been a significant increase in the number of studies conducted per year for Architecture and Design TD compared to the rest of the years. Code TD seems to have been researched throughout the years

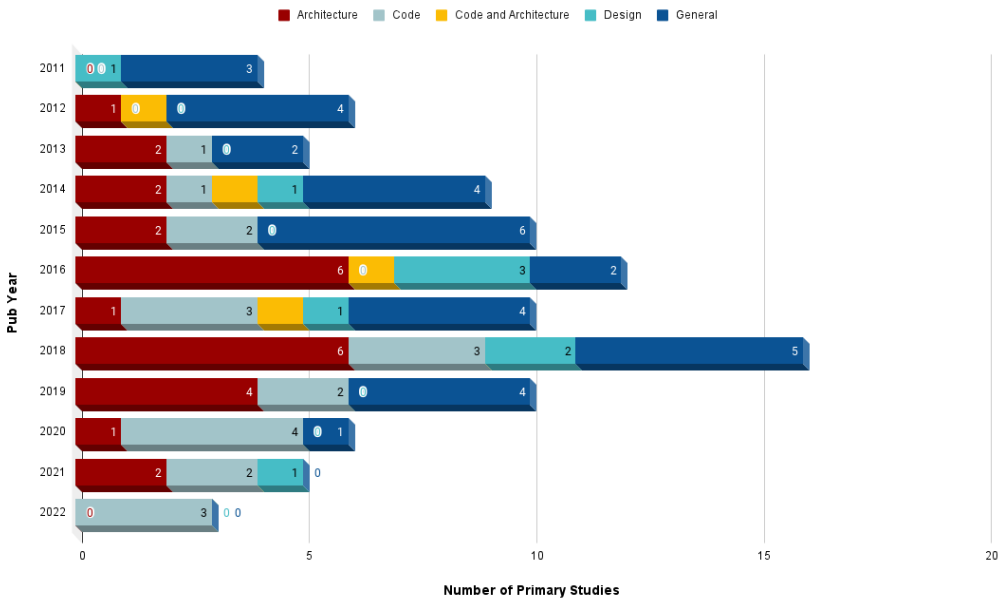

Fig. 4. Categorization of primary studies according to Publication Year

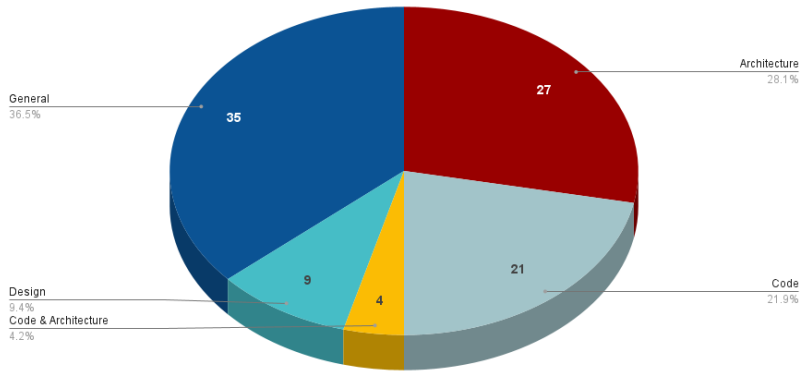

Fig. 5. Categorization of primary studies according to the type of TD

2013-2022. However, compared to Architecture and Design TD, there is no significant increase in the number of studies conducted per year for Code TD in any of the years.

3.1.2 Type of TD. Figure 5 shows the overall number and percentage of primary studies according to the different types of TD, Code, Design, and Architecture within the primary studies found in our

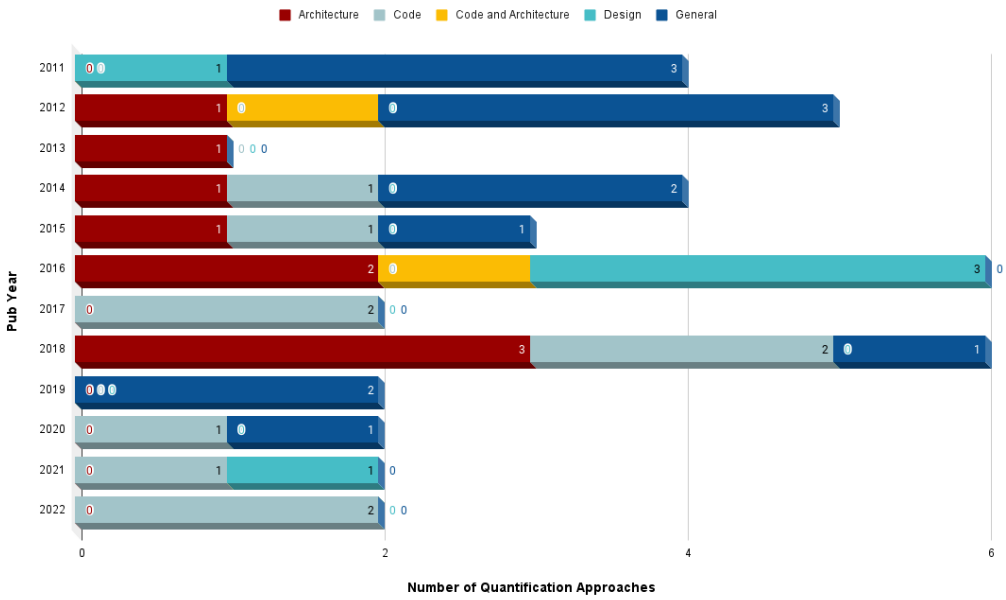

Fig. 6. Categorization of quantification approaches according to Publication Year

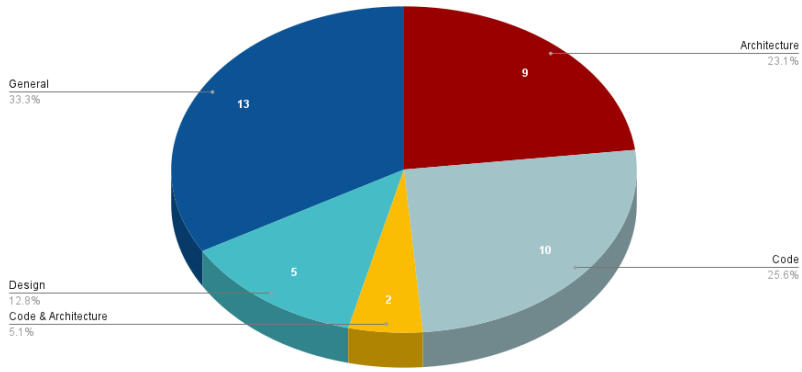

Fig. 7. Categorization of quantification approaches according to the type of TD

SMS. However, as illustrated in the Figure, most of the primary studies (35 primary studies) did not explicitly specify a type of TD. We could infer these as related to Code, Design, or Architecture TD. We labeled them as 'General.' There were 27 studies pertaining to Architecture TD, 21 pertaining to Code TD, and 9 for Design TD. There were 4 studies that discussed Code TD and Architecture TD, both in the same study. See Table 3 for descriptions of the types of TD discussed in this paper.

Other types of TD can include, for example, Requirements TD, Test TD, and Documentation TD, which we do not discuss in this paper.

Table 3. TD types discussed in this study — Definitions for Code, Design, and Architecture TD are from Alves et al. [4]

Type of TD	Description
Architecture	"Refers to the problems encountered in project architecture, for example, violation of modularity, which can affect architectural requirements (performance, robustness, among others). Normally this type of debt cannot be paid with simple interventions in the code, implying in more extensive development activities" [4]
Design	"Refers to debt that can be discovered by analyzing the source code by identifying the use of practices which violated the principles of good object-oriented design (e.g. very large or tightly coupled classes)." [4]
Code	"Refers to the problems found in the source code which can affect negatively the legibility of the code making it more difficult to be maintained. Usually, this debt can be identified by examining the source code of the project considering issues related to bad coding practices." [4]
Code and Architecture	We categorized studies that discussed both Architecture and Code TD in the same article in this category.
General	Studies that did not explicitly mention a type of TD but could be inferred as related to TD types Code, Design, or Architecture, were categorized in this category.

Summary of findings for demographics of primary studies:

- Code TD has been researched throughout the years 2013-2022.
- There is a significant increase in the number of studies conducted per year in 2016 and 2018 for Architecture and Design TD.
- Among the primary studies, General (35 studies), Architecture TD (27) and Code TD (21), are the most frequently investigated TD types.

3.2 Demographics of Quantification Approaches (RQ1)

39 unique Quantification Approaches (QAs) were identified from 96 primary studies discussing Code, Design, and Architecture TD. Table 5 in the appendix lists the QAs.

3.2.1 Publication Year. Similar to the categorization of primary studies, we categorized QAs based on their publication year (See Figure 6). We found QAs in publication years ranging from 2011 to 2022. The highest number of distinct QAs was found in 2018 and 2016, with 6 QAs in each year. In 2018, 3 QAs discussed the quantification of Architecture TD, 2 discussed Code TD, and one belonged to the General category. In 2016, 2 QAs discussed the quantification of Architecture TD, 1 discussed Code and Architecture, and 3 discussed Design TD. There have been only two distinct QAs per year since 2019, indicating that only a few distinct attempts have been made to quantify TD compared to 2018 and 2016. However, in 2018 and 2016, there were also more primary studies compared to the rest of the years.

3.2.2 Type of TD. Figure 7 shows the overall number and percentage of QAs according to the different types of TD within the dataset of QAs found in our SMS. However, as illustrated in the Figure, most of the QAs (13 of them) did not explicitly specify a type of TD. They were inferred as

related to Code, Design, or Architecture TD and therefore included. We labeled them as 'General'. 10 QAs were quantifying Code TD, 9 quantified Architecture TD, and 5 quantified Design TD and 2 of the approaches discussed the quantification of both Code TD and Architecture TD in the same article. Although more primary studies were conducted for Architecture TD compared to Code TD, the number of distinct QAs was higher by one for Code TD.

Summary of findings for demographics of QAs:

- Among the QAs found in our study, General (13 QAs), Code (10), and Architecture (9) are the most frequently investigated TD types.
- Although there were more primary studies for Architecture TD compared to Code TD, there was one additional distinct QA discussing Code TD.

3.3 Classification of TD Quantification Approaches (RQ1, RQ1.1, RQ1.2)

In this section we discuss results obtained by classifying Quantification Approaches (QAs) based on the classification scheme we developed during this study (discussed previously in Section 2.4.1 and presented in Figure 2). The classification scheme comprises the set of abstract TDQ Concepts (i.e., themes) and their high-level categories (i.e., higher-level themes): process/time, cost, benefit, probability, and priority.

The Figures in this section show a more abstract view (i.e., high-level view) of the concept mappings (D, I) and the metric mapping (M) that were discussed earlier in Section 2.4.1. — By this we mean that the figures illustrate whether a given QA discusses the abstract TDQ concept or supporting metrics for the abstract TDQ concept rather than whether the mapping was explicit, i.e., a direct mapping (D) or inferred (I). All detailed mappings D (direct), I (inferred) and M can be found in our Replication Package[1] while findings from the detailed mappings are discussed in Sections 3.4 and 3.5 and the number of mappings per mapping type is illustrated via Figure 14.

Figures 9, 10 11, and 12 illustrate the total number of QAs (via bubble size) representing each abstract TDQ concept and supporting metrics per TD type (Figures 9, and 10) and per publication year (Figures 11, and 12), respectively. Figures 8 and 13 show a more granular level of detail. They show the individual QAs pertaining to the grouping of either TD type or Publication Year. Furthermore, Figure 8 shows how many concepts are represented by each individual QA and how many QAs represent a given abstract TDQ concept regardless of the TD type and Publication year. The distribution of the QAs across the high-level themes process/time, cost, benefit, probability, and priority can be seen in all these Figures visualized by their color coding: process/time (yellow), cost (red), benefit (purple), probability (blue) and priority (green).

3.3.1 Classification grouped by TD Type. Figure 8 illustrates the classification of the 39 QAs grouped by TD types and ordered by the publication year within the block for each TD type. The 22 abstract TDQ concepts are listed in the Legend on the right side of Figure 8. The high-level themes of the abstract TDQ concepts, process/time, cost, benefit, probability, and priority, are color-coded in the cells of the classification matrix. The total number of abstract TDQ concepts discussed in each QA in our dataset can be found in the column between the classification matrix and the Legend in Figure 8 while the total count of QAs mapped to each abstract TDQ concept (regardless of the TD type and Publication year) can be found in the row below the classification matrix at the bottom of the Figure.

[1]Replication Package: https://doi.org/10.5281/zenodo.10617774

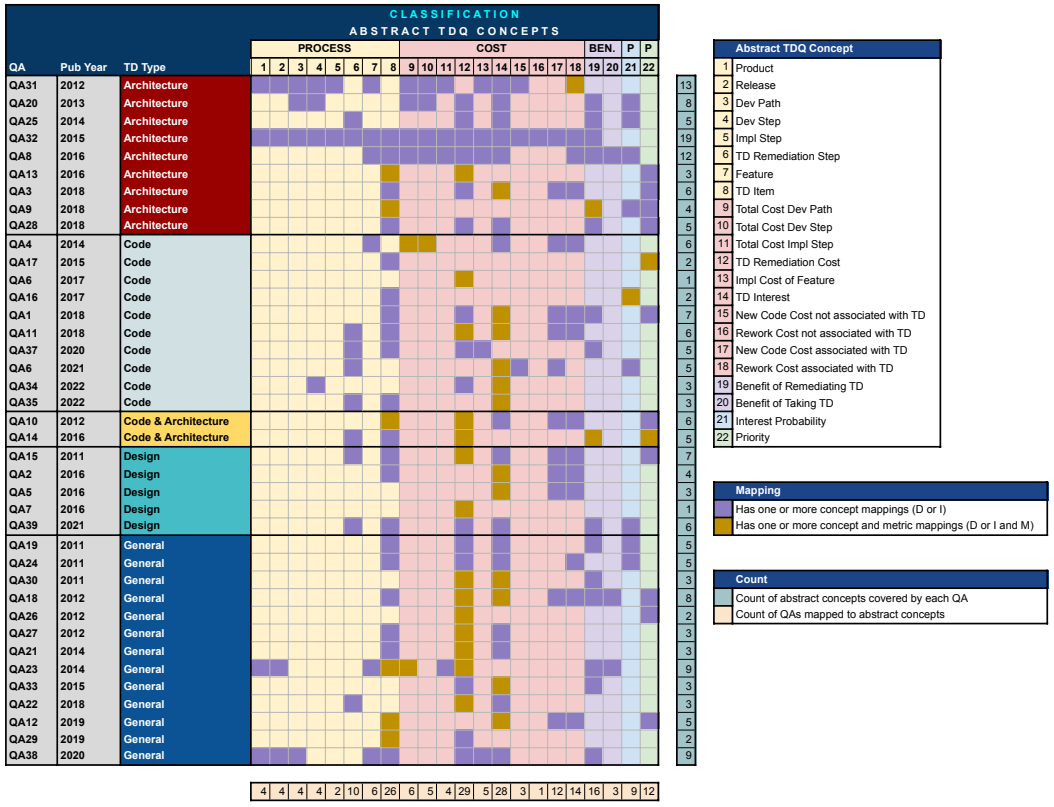

Fig. 8. Classification of Individual Quantification Approaches based on abstract TDQ Concepts, ordered by TD Type and by Publication Year within each TD Type.

The classification of QAs shows that most of the abstract TDQ concepts are represented by the QAs for Architecture TD (See Figures 8 and 9). General and Code follow. This is more evident in Figure 9 — see the row for Architecture TD; there is a bubble for every abstract TDQ concept on the x-axis.

The block of Architecture TD in Figure 8 shows the individual QAs for Architecture TD and the abstract TDQ concepts represented by each of them. For example, QA32 (notation — QA[n]), QA31, and QA8, which all belong to Architecture TD discuss 19, 13, and 12 abstract TDQ concepts, respectively (these numbers can be seen in the column between the classification matrix and the legend in Figure 8). QA32 discusses concepts 1-19. QA31 discusses concepts 1-5, 7, 9-11, 13-15, and 18 (the legend for the concepts can be seen in Figure 8 on the right side of the classification matrix). QA31 also discusses supporting metrics for quantifying the abstract TDQ concept 18, which is 'Rework Cost associated with TD.'

The abstract TDQ concepts, TD Remediation Step, TD Item, TD Remediation Cost, TD Interest, New Code Cost associated with TD, Rework Cost associated with TD, Benefit of remediating TD and Priority have been discussed among all types of TD (See Figures 8, 9). The Benefit of taking TD is discussed only among the TD types, Architecture, and General. Rework Cost not associated with TD is discussed only for type Architecture while New Code Cost not associated with TD is

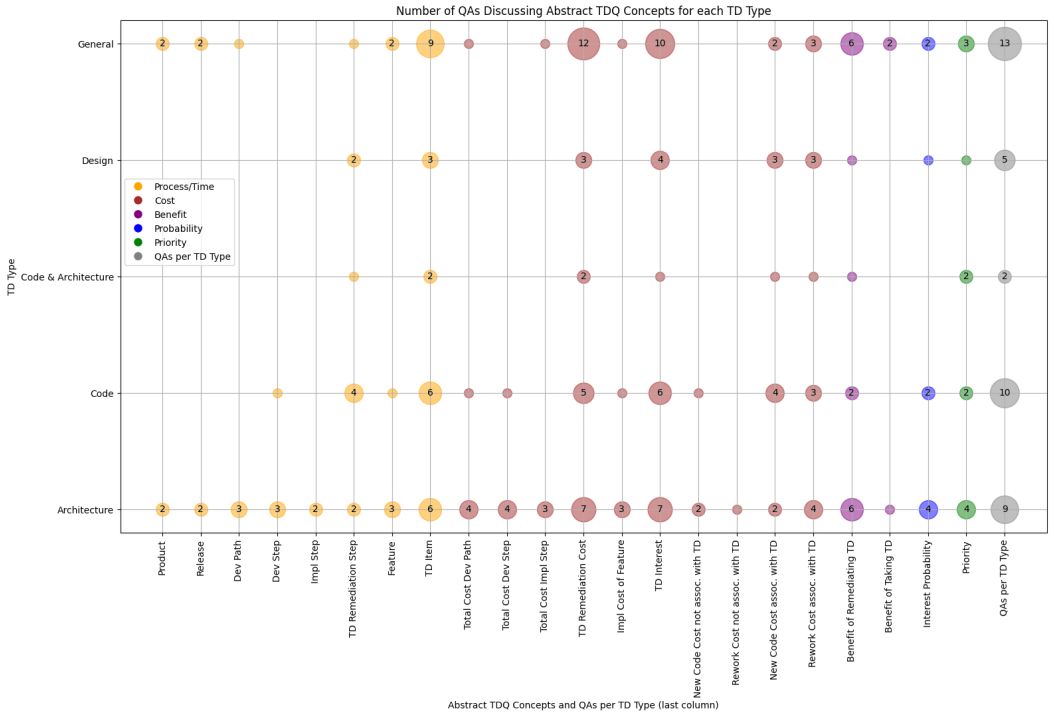

Fig. 9. Number of Quantification Approaches (QAs) discussing Abstract TDQ Concepts per TD Type. The **bubble size** shows the total number of QAs that discuss each concept per TD type.

discussed only within the types Architecture and Code. Implementation Step is discussed only within the type Code TD, which is expected.

However, costs concerning the development path, for example, the Total cost of a dev path, the Total cost of a dev step, and the Total cost of an implementation step, seem to be discussed only within the TD types, Code, Architecture, and General and not with the TD type, Design or the type Code and Architecture (See Figure 9).

Supporting Metrics have been mostly discussed for the concepts TD Remediation Cost and TD Interest for all TD types (see Figure 10). Supporting metrics for Benefit of remediating TD has been discussed for Architecture TD and the type 'Code and Architecture TD'. Supporting metrics for Rework Cost associated with TD have been discussed within Architecture TD but not among the rest of the TD types. Interest Probability has been supported by metrics within Code TD while Priority has been supported by metrics within Code and 'Code and Architecture' TD types.

Supporting metrics have not been discussed for any of the TD types to support quantifying the cost concepts New Code Cost not associated with TD, Rework Cost not associated with TD, New Code Cost associated with TD and for the process/time concepts Product, Release, Dev Path, Dev Step, Implementation Step, TD remediation step, and Feature. The process/time concepts are expected to not have supporting metrics.

In additional to the Classification of QAs among the different TD types, Figure 8 also shows the total number of abstract TDQ concepts discussed in each QA in our dataset in the column between the classification matrix and the Legend. The Figure also shows the total count of QAs mapped to

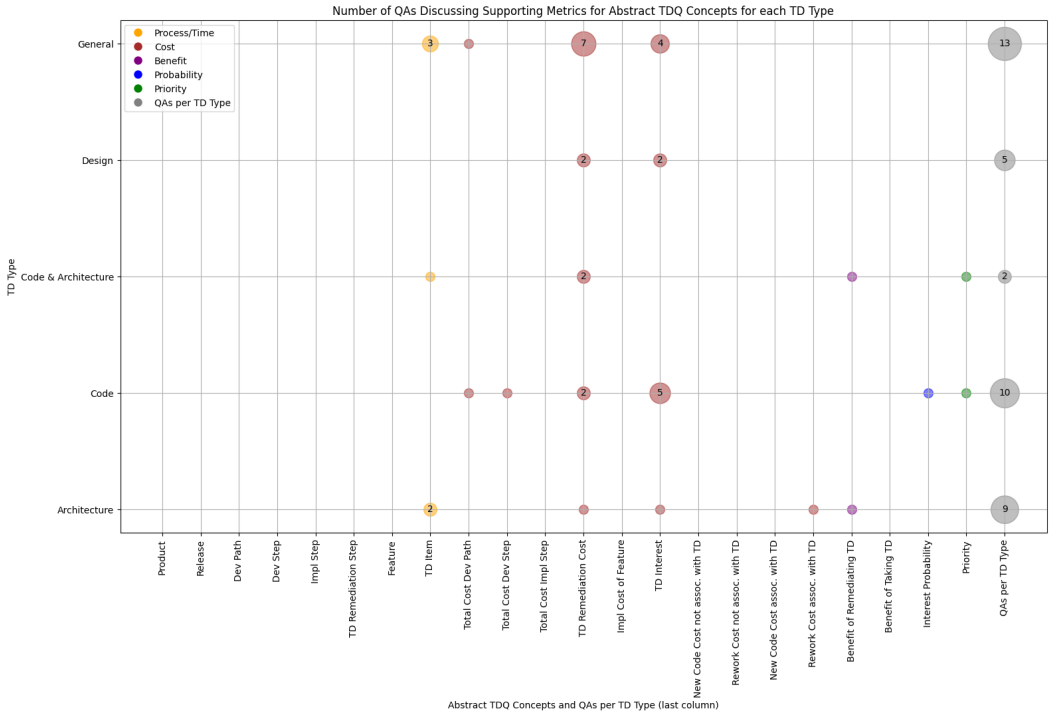

Fig. 10. Number of Quantification Approaches (QAs) discussing Supporting Metrics for each Abstract TDQ Concept per TD Type. The **bubble size** shows the total number of QAs that discuss supporting metrics for each concept per TD Type.

each abstract TDQ concept regardless of the TD type and Publication year, in the row below the classification matrix at the bottom of the Figure.

According to Figure 8, Abstract TDQ concepts, TD Remediation Cost is discussed in 29 QAs — see bottom row below the matrix, TD Interest is discussed in 28 QAs, TD Item is discussed in 26 QAs and Benefit of remediating TD is discussed in 16 QAs. These concepts are the concepts that are discussed by most of the QAs, regardless the type of TD and Publication year.

Considering the number of abstract TDQ concepts (22 in total) in the set of abstract TDQ concepts (see Legend in Figure 8) and the count of QAs representing each concept (see bottom of Figure 8), an observation that could be made is that quantification approaches are discussing more cost concepts compared to benefit concepts regardless of the TD type or publication year. There are 10 cost concepts and only 2 benefit concepts in the set of abstract TDQ concepts, and the counts for QAs are comparatively higher for cost concepts than for benefit concepts (See Figures 8 and 9). Similarly, counts for concepts supported by metrics are comparatively higher for cost concepts than for benefit concepts (See Figures 8 and 10).

3.3.2 Classification grouped by Publication Year. Figure 11 shows which abstract TDQ concepts have been discussed over the publication years. Most of the abstract TDQ Concepts, such as TD Remediation Cost, TD Interest, and TD Item, have been discussed over most of the publication years.

However, an observation made here is that the TDQ concepts, Total cost of a dev path, the total cost of a dev step, and the total cost of an implementation step have been discussed between 2012 -

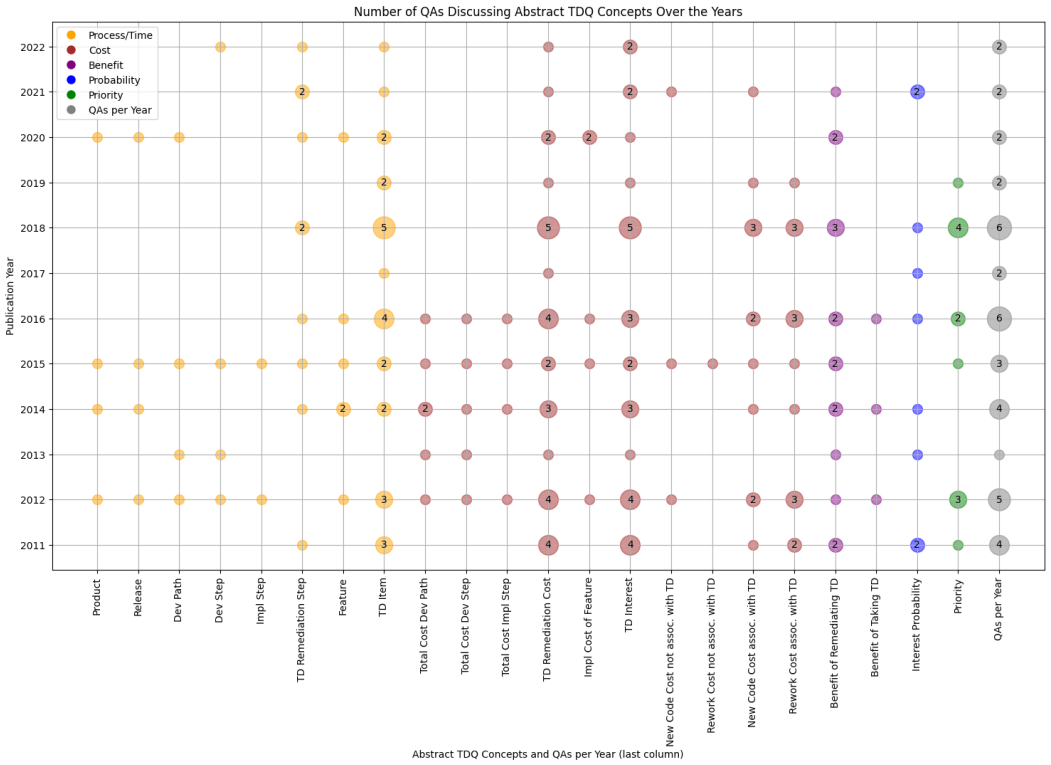

Fig. 11. Number of Quantification Approaches (QAs) discussing Abstract TDQ concepts over the Publication Years. The **bubble size** shows the total number of QAs that discuss each concept per Publication Year.

2016 but not before or after. The process/time related concepts Product, Release, Dev Path, Dev Step, and Implementation Step have been discussed but not continuously. This is similar with New Code Cost not associated with TD and Rework Cost not associated with TD.

Another observation is that the Benefit of taking TD has been discussed in 2012, 2014, and 2015 but not in the rest of the publication years. The benefit of Remediating TD has been discussed over most of the publication years except 2017, 2019 and 2022. However, supporting metrics have been proposed for this concept only in 2016 and 2018 (see Figure 12).

Supporting metrics are discussed frequently for TD remediation Cost and TD Interest as well as some for TD Item over most of the publication years (see Figure 12) while supporting metrics for quantifying Rework Cost associated with TD (2012), Interest probability (2017) and Priority (2015, 2016) have some occurrences.

Figure 13 shows the same information as Figure 8 except ordered by Publication Year. It shows which individual QAs are relevant to each publication year, adding a granular level of detail to Figure 11 where the total number of QAs (bubble size) for each abstract TDQ concept per publication year is illustrated. For example, it shows which individual QAs represent the abstract TDQ concepts for the years 2012 and 2015, where most of the abstract TDQ Concepts have been discussed. However, it is evident from this Figure that QA31 and QA32 contribute to this by discussing most of the abstract TDQ concepts.

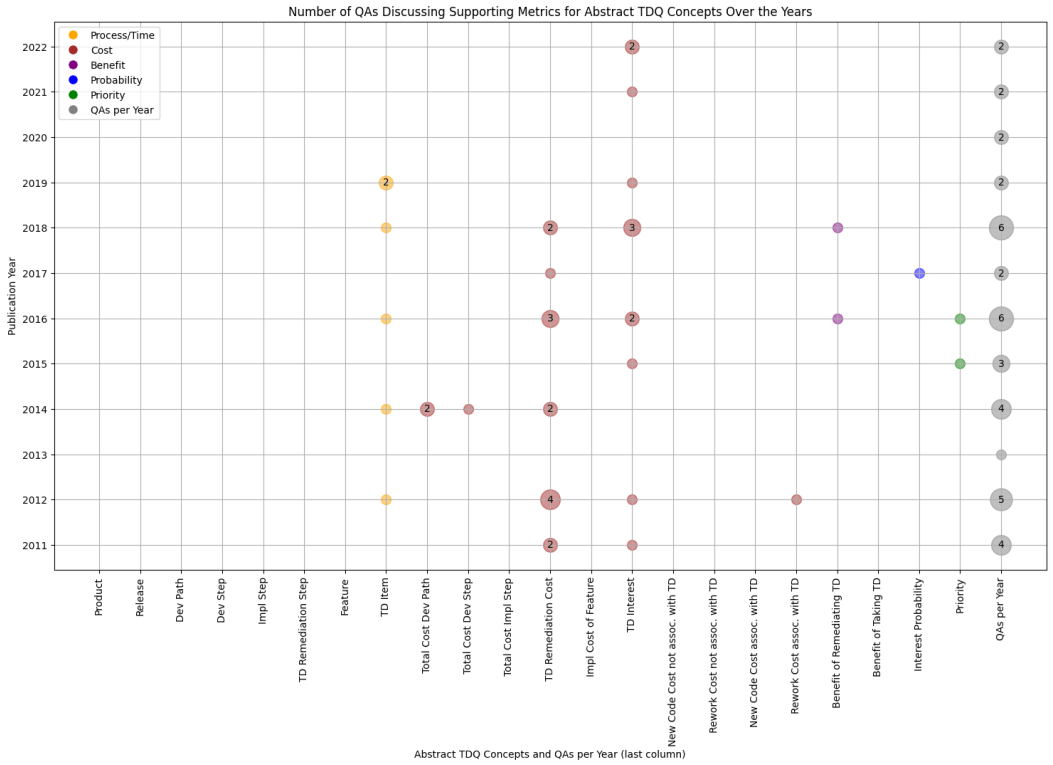

Fig. 12. Number of Quantification Approaches (QAs) discussing Supporting Metrics for each Abstract TDQ concept over the Publication Years. The **bubble size** shows the total number of QAs that discuss supporting metrics for each concept per Publication Year.

Summary of findings from the Classification of QAs (RQ1, RQ1.1, RQ1.2):

- QAs for Architecture TD discuss all 22 abstract TDQ concepts collectively.
- TD Item, TD Remediation cost, TD Interest, and Benefit of Remediating TD are the concepts most frequently represented among the QAs regardless of the type of TD and the publication year. Supporting metrics are most frequently discussed for TD Remediation cost and TD Interest.
- Cost concepts are more frequently investigated than benefit concepts. Metrics supporting the quantification of benefits are also fewer than for costs.
- Costs concerning the development path are not investigated for Design TD.
- The Benefit of remediating TD has been discussed more frequently compared to the Benefit of taking TD. However, supporting metrics have been proposed only in 2016 and 2018.
- New Code and Rework Costs associated with TD have been discussed among all TD Types. However, supporting metrics are discussed only for Rework costs associated with TD and only for Architecture TD.

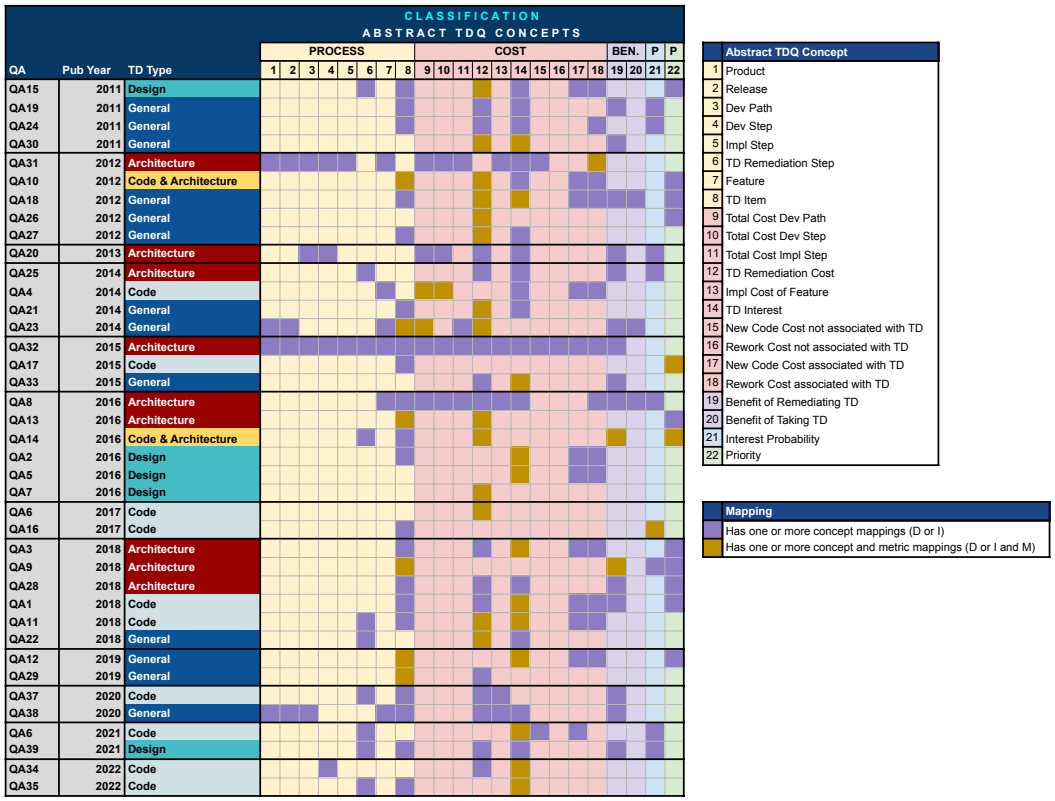

Fig. 13. Classification of Individual Quantification Approaches based on abstract TDQ Concepts, ordered by Publication Year

3.4 TDQ Concepts (RQ 1.1)

Figure 14 illustrates the counts of the detailed concept and metric mappings (D, I, M) between the concepts extracted from QAs and the abstract TDQ Concepts regardless of the TD type or Publication Year. That is, how many QAs have discussed a particular concept and have proposed supporting metrics, and how many concepts in total have been mapped from the QAs to the abstract TDQ Concepts. The size of the bubble shows the count of mappings. The y-axis shows the number of QAs representing an abstract TDQ concept. The x-axis shows the abstract TDQ concepts.

Sections 3.4.1 and 3.4.2 below discuss the most frequently discussed, and the least frequently discussed TDQ concepts in terms of the detailed mappings, D and I. Mapping M is discussed in Section 3.5. Tables 6, 7, and 8 in the Appendix list the concepts and metrics from the QAs that were mapped to the abstract TDQ concepts.

3.4.1 Most frequently discussed TDQ Concepts. Below we discuss results obtained for the most frequently discussed abstract TDQ Concepts with respect to the concept mappings D and I.

Mapping D — According to the obtained results, TD Item, Remediation Cost, TD Interest, and Benefit of Remediating TD appear to be the most frequently discussed concepts (see highest counts in Figure 14). 19, 24, 23 and 9 QAs (y-axis) discuss these concepts while, 19, 25, 24 and 11 concepts from the QAs (bubble size) are mapped to these abstract TDQ concepts.

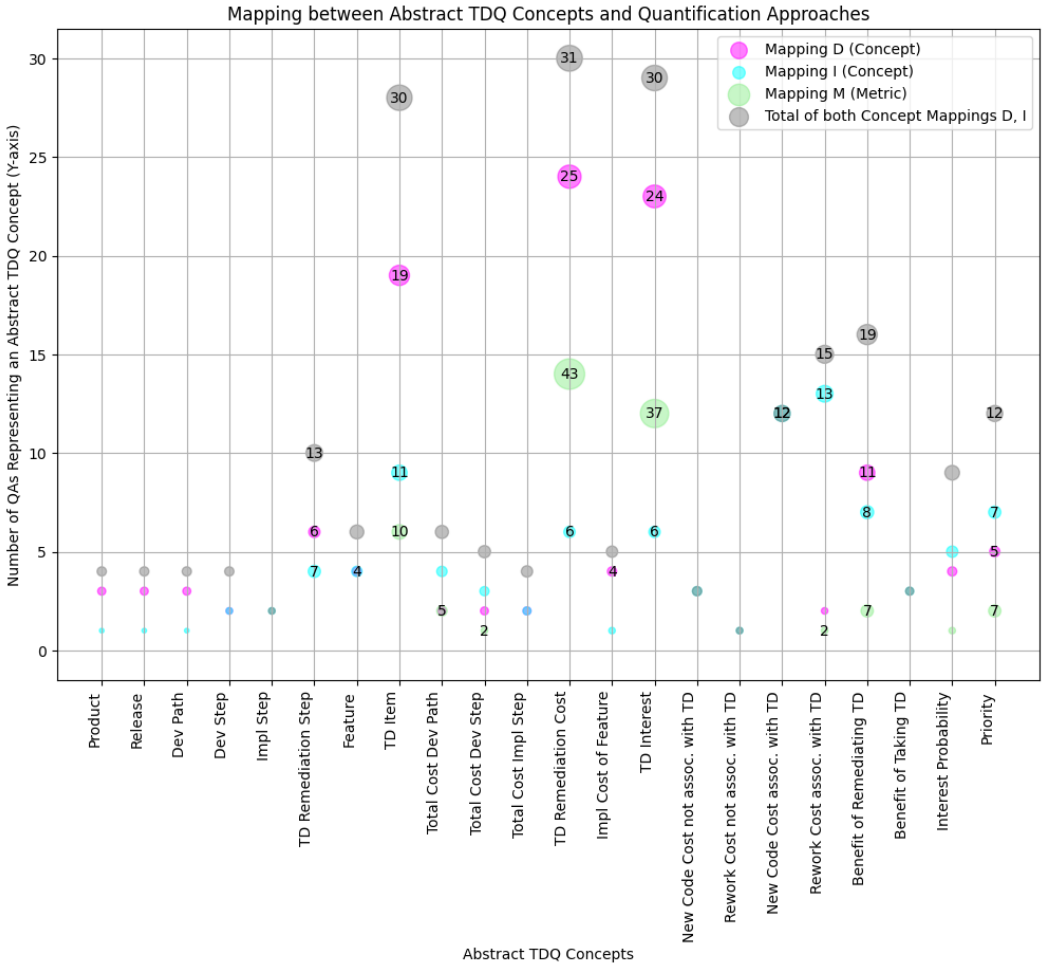

Fig. 14. Mappings between Abstract TDQ Concepts and TD Quantification Approaches. The **bubble size** shows how many concepts from each QA were mapped to an abstract TDQ concept.The legend describes the color coding of the bubbles for the different mapping types.

Mapping I — According to the obtained results, TD Item, New Code Cost associated with TD, and Rework Cost associated with TD appear to be the most frequently discussed concepts (see highest counts in Figure 14). 9, 12, and 13 approaches discuss these concepts (y-axis) while 11, 12, and 13 concepts from the QAs are mapped to the abstract concepts (bubble size).

As discussed previously in Section 2, the two different mappings D and I indicate that the concepts were either straightforward to extract or inferred (i.e., derived). However, when we combine the results obtained for both types of mappings, it is evident that *TD item, Remediation cost, TD interest and Benefit of remediating TD* receive the highest number of mappings 30, 31, 30, and 19, respectively. At least 12 (12 QAs for New code cost associated with TD) and at most 30 (30 QAs for TD Remediation cost) out of 39 QAs are mapped to these frequently discussed abstract TDQ concepts (TD item, Remediation cost, TD interest and Benefit of remediating TD), indicating that the majority of the QAs in our SMS discuss similar aspects of TD quantification.

3.4.2 Least frequently discussed TDQ Concepts. Below we discuss results obtained for the least frequently discussed TDQ Concepts with respect to the concept mappings D and I (See Figure 14).

Mapping D — According to the obtained results, some abstract TDQ concepts received only a few explicit (D) mappings. Dev Step, Dev Path total Cost, Dev Step Total Cost, Impl Step Total Cost, and Rework Cost associated with TD appeared to be the least frequently discussed concepts. Two QAs (y-axis) each and 2, 3, 3, 3, and 2 concepts (bubble size) from the QAs were mapped to those abstract TDQ concepts. There were also a few concepts that received zero D mappings. They were Impl Step, New Code Cost not associated with TD, Rework cost not associated with TD, New Code Cost associated with TD, and Benefit of taking TD.

Mapping I — According to the obtained results, Product, Release, Dev Path, Dev Step, Impl Step, Impl Step Total Cost, Impl Cost of Feature, and Rework Cost not associated with TD appeared to be the least frequently discussed concepts for Mapping I. For concepts Product, Release, Dev Path, Impl Cost of feature, and Rework Cost not associated with TD, one QA each, and for concepts Dev Step, Impl Step, and Total cost of Impl, two QAs each discuss these concepts. The number of concepts mapped from QAs (see y-axis), were 1, 1, 1, 2, 2, 3, 2, and 2 for the abstract TDQ concepts in the order of occurrence (bubble size) in Figure 14.

However, the observation here is that there has been at least one inferred mapping (mapping I) to the abstract TDQ concepts, even for instances with zero explicit mappings, e.g., for the abstract TDQ concepts, Impl Step and Rework cost not associated with TD. This was also evident once the results obtained for both types of mappings were combined. — There were no abstract TDQ concepts with zero mappings. Concepts that had zero D mappings, Impl Step, New Code Cost not associated with TD, Rework Cost not associated with TD, New Code Cost associated with TD, and Benefit of taking TD still had a count in I mappings.

Another observation made by examining the total mappings for D and I is although the abstract TDQ concepts New Code Cost associated with TD and Rework Cost associated with TD had fewer explicit (D) mappings, they had a considerable count of inferred (I) mappings. The total mapping for these two concepts was derived as; 12 and 15 QAs (y-axis), and 12 and 15 approach concepts (bubble size) were mapped in total for the two abstract TDQ concepts.

Summary of findings for TDQ Concepts (RQ1.1):

- There were no abstract TDQ Concepts that had zero mappings, i.e., all QAs discussed the quantification of at least one abstract TDQ concept when the total of D and I mappings were considered in order to determine what a QA quantifies.
- TD Remediation Step, TD Item, TD Remediation Cost, TD Interest, New code cost associated with TD, Rework cost associated with TD, Benefit of remediating TD and Priority, received the highest number of concept mappings considering D and I mappings in total, indicating that most of the QAs found in our SMS discuss these aspects of TD quantification similarly. *(TD Item, TD Remediation Cost, TD Interest, and Benefit of remediating TD are the top four.)*
- Although New Code Cost associated with TD and Rework Cost associated with TD had zero or fewer explicit (D) mappings, they had a considerable amount of inferred (I) mappings. Therefore, these concepts are worthwhile for further investigation.
- Impl Step, New Code Cost and Rework Cost not associated with TD, New Code Cost associated with TD, and Benefit of taking TD still had a count in inferred (I) mappings although they had zero direct (D) mappings. They could also be further investigated.

3.5 Metrics supporting the Quantification of TDQ Concepts (RQ 1.2)

Bubbles for the Metric mapping in Figure 14 show the number of QAs discussing some form of measurement for an abstract TDQ concept (y-axis) and how many metrics in total from the approaches map to a given abstract TDQ Concept (bubble size). According to the obtained results, *TD Item, Remediation Cost and TD Interest* appear to be the most frequently discussed concepts in terms of the concepts where some form of measurement has been discussed. 6, 14 and 12 QAs discuss metrics for these concepts while 10, 43, and 37 metrics from the approaches in total are mapped to the abstract TDQ concepts. It seems that most of the QAs provide metrics to quantify Remediation Cost and TD Interest (Remediation Cost being the most frequently discussed one), as well as provide some form of measurement for TD Items as well.

However, there are concepts that have no form of measurement provided for them in the QAs (zero counts). These include; the process/time-related concepts (except TD Item), a few of the Cost concepts (Total cost of an implementation step, Implementation Cost of a Feature, New code cost not associated with TD, Rework Cost not associated with TD, and New Code Cost associated with TD), and one benefit concept, the Benefit of Taking TD. It is expected that process/time-related concepts will have no measurement associated with them. However, it is interesting that some of the cost-related concepts, and especially the Benefit of taking TD, have no metrics associated with them.

Summary of findings for Metrics supporting the quantification of Concepts (RQ1.2):

- TD Item, Remediation Cost, and TD Interest are the most frequently discussed concepts in terms of the concepts where some form of measurement has been discussed.
- Some cost concepts have not been adequately supported by metrics, e.g., Total cost of Impl Step, New Code cost associated with TD, New Code cost not associated with TD, and Rework cost not associated with TD.
- The quantification of the Benefit of taking TD has not been supported by metrics, although the concept has been discussed in some QAs.

4 DISCUSSION

In this section, we first discuss the implications of the findings from the analysis of TD quantification approaches for TD types Code, Design, and Architecture found in our SMS in Section 4.1. Later, in Section 4.2, we discuss potential future research directions, and in Section 4.3, we discuss implications for practitioners.

4.1 Implications of Findings

4.1.1 Quantifying TD at the 'code' level. Our findings suggest that most quantification approaches are still focusing on the quantification of 'Code TD' compared to other types of TD. Code TD captures TD at a more granular lower level, while Design TD and Architecture TD capture TD at higher levels in software product architecture. Although there have been more primary studies for Architecture TD, the number of quantification approaches is slightly higher for Code TD. However, quantification approaches for Architecture TD collectively cover all 22 abstract TDQ concepts. This indicates that the overall bigger picture is explored in approaches quantifying Architecture TD. This suggests the opportunity to investigate the quantification of types of TD exploring TD at a higher level, and how they could impact the overall software product.

4.1.2 Current focus is on the cost concepts. The quantification approaches found in our SMS focus mainly on TD remediation Cost, TD Interest, TD Item, and Benefit of Refactoring. Out of these concepts, TD Remediation Cost and TD Interest are the concepts that are well supported by metrics — these are both cost concepts. This indicates that the current focus is on cost concepts, and the benefit concepts are not adequately considered in TD quantification. This raises the question of whether cost concepts alone can make well-informed decisions for TDM. According to Li et al.'s definition of TD measurement: "Quantifies the benefit and cost of known TD in a software system through estimation techniques, or estimates the level of the overall TD in a system" [39], we should also focus on benefit concepts. Therefore, we should critically assess the ability to make well-informed TDM decisions using quantification approaches considering both cost and benefit concepts.

4.1.3 Remediating TD in retrospect. An observation we made through the analysis of the quantification approaches is that they mainly focus on 'remediating TD,' the cost to remediate TD, and the benefit of remediating TD. This indicates that TD is dealt with after the fact, i.e., in retrospect, when TD already exists in the software product. The quantification approaches rarely discuss the potential benefit of taking on TD strategically, for example, to gain a competitive advantage. The lack of supporting metrics to measure the benefit concepts shows that the metaphor 'Technical Debt' is not used as intended. The financial metaphor 'debt' suggests taking a loan to make progress on some task while paying off the interest or making a one-off payment whenever it still yields the benefits of taking the loan in the first place. This is a prudent, deliberate action. However, our findings suggest a lack of attention in the current quantification approaches to the fact that TD can be used strategically during software development.

4.1.4 New Code and Rework Costs. There can be New Code and Rework costs associated with TD as well as not associated with TD. — costs that are a consequence of TD and costs that are not a consequence of TD.

An observation we made with respect to these costs is that New Code Costs and Rework Costs associated with TD were represented in all the TD types analyzed in this paper, while Rework costs not associated with TD is discussed only with respect to Architecture TD and New Code costs not associated with TD is discussed only with respect to Architecture and Code TD. However, of all four concepts, supporting metrics could be found only for Rework costs associated with TD and only for Architecture TD. This suggests that rework is potentially occurring more frequently with respect to the software architecture and is more likely to be measured as a consequence of TD in the context of Architecture TD. However, this does not mean that these costs cannot be measured for other types of TD.

4.1.5 Understanding the quantification of TD. There were no abstract TDQ concepts with zero mappings. This suggests that the developed set of abstract TDQ concepts sufficiently captures what is required to understand the quantification of TD and can be considered a unified representation of the quantification approaches. However, we are yet to understand how useful these concepts are in TDM decision-making. For this, we suggest that the next step should be to better understand the relationships between these abstract TDQ concepts. We plan to do this in our future work.

4.2 Future Research Directions

4.2.1 Addressing the lack of focus on TD quantification in the recent research literature. Our findings indicate a lack of emphasis on TD quantification in recent years (since 2019). To address this gap, we encourage researchers to prioritize the investigation of TD quantification since TD quantification is a crucial TD Management (TDM) activity that can support the decision-making process. Quantifying

TD can support making informed decisions regarding TDM, such as taking on TD to gain a competitive advantage and remediating it prudently while the benefit is still yielded.

4.2.2 Investigating the quantification of TD types where TD quantification is less-explored. Demographics of QAs revealed a predominant focus on TD types: Code, Architecture, and General. We encourage researchers to investigate the quantification of other types of TD, such as Design TD, where TD quantification has been less explored.

4.2.3 Exploring insufficiently investigated TDQ Concepts. Classifying the QAs based on our concept-based classification scheme revealed gaps in the research literature where some TDQ concepts were not adequately investigated. We encourage researchers to investigate these concepts and metrics that could support quantifying such concepts.

One such example concept is the *Benefit of taking TD*. Despite the potential benefits of the strategic adoption of TD (e..g., to gain a competitive advantage) this concept has received very limited attention with only 3 QAs discussing the concept and only three concepts mapped from the 3 QAs. The concept has not been supported with any form of measurement, indicating a lack of supporting metrics. Therefore, we encourage researchers to address this gap by examining this concept and potential metrics for its quantification. Furthermore, the benefit of taking TD has been discussed only among Architecture and General types so far. This indicates the need for exploring the concept with respect to the different types of TD.

Total Cost of Implementation Step, New Code cost, and Rework Costs not associated with TD are similarly insufficiently explored concepts. Rework costs not associated with TD are discussed only for Architecture TD. Although these costs represent costs that may be unaffected by TD, they may play a role in TDM decision-making, for example, with respect to allocating the time spent by developers on managing TD. Therefore, investigating the role of these concepts in TDM decision-making can be worthwhile. However, the Total Cost of Implementation Step may still be affected by the TD Interest. Therefore, investigating the relationship between these concepts is encouraged.

4.2.4 Addressing the insufficient exploration of benefit concepts. Cost concepts are frequently investigated in the existing literature for TD quantification. Although various metrics have supported the quantification of many cost concepts, metrics supporting the quantification of benefit concepts are fewer than for cost concepts. To address this research gap, we encourage researchers to focus on developing suitable metrics to quantify the benefit concepts effectively. There are two benefit concepts in the set of abstract TDQ concepts, *the benefit of taking TD* and *the benefit of remediating TD*. However, compared to the benefit of taking TD, the benefit of remediating TD has been discussed more frequently as a concept. Yet, supporting metrics have been proposed only in 2016 and 2018. Supporting metrics have not at all been discussed for the benefit of taking TD.

It is worthwhile to investigate the possibility of making better-informed decisions regarding TD management if the quantification of the benefit concepts is known. For example, it may be worth exploring whether stakeholders would make better decisions if the benefit of taking TD is known in a given situation and not only the cost of remediating.

4.2.5 Further investigation of New Code Costs and Rework Costs associated with TD. Although *New code cost associated with TD* and *Rework cost associated with TD* (i.e., the costs affected by TD — descriptions can be found in Table 2) had zero or few explicit (D) mappings from the QAs, they had multiple inferred (I) mappings (12 and 15 QAs, 12 and 15 concepts mapped from QAs). This suggests that these concepts would benefit from further investigation to better understand their implications and to develop appropriate metrics for their quantification.

4.2.6 Addressing the insufficient exploration of metrics to support the quantification of TDQ concepts. Some cost concepts have not been adequately supported by metrics, e.g., Total cost of Impl Step, New Code cost associated with TD, New Code cost not associated with TD, and Rework cost not associated with TD. Costs concerning the development path are not investigated in particular for Design TD. Addressing these gaps in metric support can enhance the quantification of TD.

4.2.7 Quantifying TD in a timely manner. As discussed in the implications of findings (Section 4.1), an observation made with respect to the current literature is that most of the existing TD quantification approaches were quantifying TD retrospectively to remediate accumulated TD. Another aspect of TD is to use it for strategic reasons, for example, to gain a competitive advantage. However, for this, TD must be quantified during software development. We encourage researchers to develop quantification approaches supporting this aspect of TD management.

4.2.8 Investigating potential aspects of TD quantification for well-informed TDM decision-making. TD Item, Remediation Cost, TD Interest, and Benefit of remediating TD received the highest number of concept mappings, indicating that most of the QAs found in our SMS consider these aspects the most significant aspects of TD quantification. However, the question is whether these aspects of TD quantification alone can make well-informed decisions, i.e., if they lead to a reliable TD quantification useful for TDM decision-making. Researchers should critically asses if these aspects are sufficient to make well-informed decisions and explore other potential aspects that could enhance the decision-making process to enable more effective TD management in practice. The set of abstract TDQ concepts developed in our study serves as a reference for identifying such potential aspects.

4.3 Implications for Practitioners

We provide a classification of Quantification Approaches (QA) classified based on a set of abstract TDQ concepts and their high-level themes process/time , cost, benefit, probability and priority. The classification also shows which abstract TDQ concepts are supported by metrics for each QA. See Figure 8.

Practitioners may want to decide what existing quantification approaches are fit for their purpose, depending on their particular needs. They could use our classification map of QAs to select a QA for their particular need.

As an example, if they want to make decisions if and when to fix TD by considering the TD Remediation Cost for a given type of TD such as Architectural TD, they may want to select one of the QAs representing these concepts. QAs 20, 25, 32, 8, 13, 3, and 28 all which quantify Architecture TD represent TD remediation cost. However, only QA13 discusses supporting metrics for TD remediation cost out of the Architecture TD QAs discussing TD remediation cost. Therefore, if practitioners are searching for an existing approach that already discusses metrics that support the quantification they want to perform (TD remediation cost in this example), they may want to select QA13 from the QAs given in our classification as fit for their purpose.

5 THREATS TO VALIDITY

In this section, we discuss threats to the validity of our study based on the guidelines provided by Petersen et al. [50].

5.1 Descriptive Validity

Descriptive Validity refers to how a study describes observations accurately and objectively. To reduce this threat, we recorded the data (Quantification approach ID, study title, publication year, TD type, and concepts extracted from the quantification approach) in a tabular format. Doing so

objectified the data extraction process. Metadata such as the study title and the publication year was automatically recorded from the searched databases.

Furthermore, the first author objectively extracted TDQ concepts from the quantification approaches and coded them systematically following the process described in Section 2.4. The other authors checked the concepts extracted from the quantification approaches (i.e., initial codes), resulting themes (i.e., abstract TDQ concepts), and high-level themes (i.e., process/ time, cost, benefit, probability, and priority) during multiple iterative meetings held during data extraction and analysis. Any concerns brought up by the other authors were discussed and resolved during the iterative consensus meetings. The classification scheme (See Figure 2) was discussed and agreed upon among all authors prior to analyzing the quantification approaches for reporting results.

5.2 Theoretical Validity

Theoretical Validity refers to our ability to capture what we intend to capture. This applies to identifying primary studies, identifying quantification approaches, data extraction, and classification in our study.

5.2.1 Identification of Primary Studies/ Sampling. Primary studies could have been missed during the search process. To mitigate this threat, we developed our search string in multiple iterations before finalizing a satisfactory one. We piloted the search string multiple times with one of the major databases, SCOPUS. Then, we evaluated our search results with the reference set of articles obtained from the TechDebt conference proceedings in 2018 and 2019, as listed on their website [1]. All the articles from the reference set were found in SCOPUS search results with the final search string. Hence, the search string was verified that it captures what we intend to capture.

To expand the possibility of including all relevant articles in our sample, we developed our search string by using keywords and their synonyms and wildcards (*) to capture possible variations of the keywords, for example, plurals and verb conjugations. We applied the search query to the title, abstract, and keywords to increase the probability of finding all relevant articles. Furthermore, we did not limit our search to a particular period. The search string was discussed and agreed among all authors prior to conducting the search.

We conducted backward reference snowballing as a way to complement the database search so that articles that may have not been captured by the search string could be found during snowballing. However, we did not conduct forward reference snowballing when selecting the set of primary studies. Yet, we evaluated the validity of our findings by conducting forward reference snowballing on the study QA32 [32], which had the highest number of mappings for the set of abstract TDQ concepts. Only two papers resulted from this. This indicates that the conclusions of this study are not likely to change.

The inclusion and exclusion criteria were evaluated with the reference set of articles from TechDebt [1] before screening articles for inclusion. The inclusion and exclusion criteria and the selected studies were discussed and agreed upon among all authors before finalizing them as the sample.

5.2.2 Identification of Quantification Approaches. We identified quantification approaches from primary studies based on our definition for a quantification approach; *"An approach that discusses concepts and metrics that could support TDM decision-making"*. Furthermore, while identifying unique quantification approaches, we grouped studies that discussed the same approach and selected the article that discussed quantification concepts. We acknowledge that this could introduce researcher

[1]Papers accepted in Tech Debt 2018, 2019: https://2018.techdebtconf.org/#event-overview, https://2019.techdebtconf.org/#Accepted-Papers

bias. However, we think we sufficiently mitigated this threat by discussing the selection of the quantification approaches among the researchers in multiple iterative consensus meetings where disagreements were resolved.

5.2.3 Data Extraction and Classification. The first author performed the analysis of the selected quantification approaches. The coding process was systematic and strictly adhered to. The other authors checked the concepts extracted from the quantification approaches (i.e., initial codes), resulting themes (i.e., abstract TDQ concepts), and high-level themes (i.e., process/ time, cost, benefit, probability, and priority) during multiple iterative consensus meetings.

The coding process did not add new concepts to the set of abstract TDQ concepts after concepts from 33 out of 39 approaches were already mapped to the abstract TDQ concepts. The classification scheme (Figure 2) was also discussed and agreed upon among all authors prior to finalizing it. The classification scheme was used as the framework to perform the analysis to find answers to our research questions, leading to interesting implications of the findings and future research directions for the research community as well as implications for practitioners.

5.3 Generalizability

Generalizability refers to the internal and external generalizability of the study. Systematic mapping studies typically follow a common process. For example, researchers usually follow the guidelines by Petersen et al. [50]. We followed the same guidelines. Hence, the threat to internal generalizability is mitigated.

In terms of external generalizability, we do not claim our classification scheme or the set of abstract TDQ concepts to be complete. However, since all quantification approaches in our dataset could be represented by the set of abstract TDQ concepts and their high-level themes, we can conclude that the set of abstract TDQ concepts sufficiently captures the concepts required to discuss TD Quantification for Code, Design, and Architecture TD.

5.4 Interpretive Validity

Interpretive Validity maps to conclusion validity. The conclusions should be reasonable given the data [50]. Researcher bias could apply to interpreting the data since the first author drew the conclusions of our study. However, in our case, the data was interpreted based on the set of abstract TDQ concepts and their high-level themes (process/time, cost, benefit, probability, priority) that were systematically developed (see Section 2) during our study. We discussed the results, their interpretations, and the conclusions among all authors and agreed on them prior to reporting them in this paper.

5.5 Repeatability

The detailed reporting of the research process preserves the repeatability of the study. We have reported the systematic mapping process that we followed in Section 2 and provided a Replication Package[1] as supplementary material. Furthermore, the process we followed applied the guidelines by Petersen et al. in [50], a commonly followed process for SMSs by the research community.

6 RELATED WORK

In this Section we discuss related work firstly in terms of secondary and tertiary studies in Section 6.1 and secondly in terms of other related work in Section 6.2.

[1]Replication Package: https://doi.org/10.5281/zenodo.10617774

6.1 Secondary and Tertiary Studies

Several secondary and tertiary studies related to TD have been published since 2012. However, the topic of *'TD Quantification'* or measurement remains under-explored. The phenomenon is especially not explored conceptually, i.e., a study that surveys TD quantification conceptually is missing in the existing surveys. We fill this gap by conducting a systematic mapping study that focuses on TD quantification, conceptually.

In our mapping study, we emphasize the importance of understanding TD quantification conceptually as an important TDM activity since it supports other TDM activities such as *'prioritization'* and *'repayment'* [13, 41]. We analyzed concepts and metrics discussed in TD quantification approaches to identify gaps in terms of what is quantified in the existing quantification approaches based on a set of abstract TD quantification concepts (and their high-level themes process/time, cost, benefit, probability, and priority) that sufficiently capture TD quantification conceptually. Implications of findings and recommendations for researchers and practitioners were discussed in Sections 4.1, 4.2, and 4.3.

Although most of the secondary and tertiary studies have focused on aspects different from TD quantification, we discuss them below and draw connections to our study where possible. However, a few studies focus on TD measurement tools, which are discussed under the focus area 'Measurement (quantification) of TD' in Section 6.1.3. Table 4 summarizes the contributions of all the studies discussed in this section, along with the contributions of our mapping study.

6.1.1 Concept of TD. Tom et al. [60] were the first to conduct a secondary study on TD in 2012. They focused on the concept of TD, attempting to provide a holistic view of TD. The outcome was a theoretical framework comprising TD dimensions, attributes, precedents, and outcomes. Alves et al. [4] proposed an ontology of terms on TD in their systematic literature review. Different types of TD and their indicators were identified during this study. Alves et al. [3] provided an improved version of the ontology of terms on TD proposed by the same authors in 2014. They provided a list of indicators to identify TD, a list of TD management strategies, data sources used in TD identification activities, and software visualization techniques used to identify and manage TD.

6.1.2 TD Management (TDM), TDM Tools and Strategies. Li et al. [39] conducted their study on TD and TDM. They classified TD into ten types and TDM activities into eight types. Twenty-nine tools for TDM were identified. The authors emphasized the need for tools for managing the different TD types during the TDM process. According to the authors, code-related TD and its management had gained the most attention at the time of conducting their study. However, this has not changed since then. We emphasize the need to investigate other types of TD and their quantification since the current focus still remains on code-related TD. Li et al. [39] identify the lack of an underlying theory and models to support TD identification and measurement. Our work fills this gap by establishing a theoretical foundation identifying a consolidated set of abstract TDQ concepts and high-level themes sufficient to capture TD quantification conceptually.

Rios et al. [53] conducted a tertiary study in 2018. They evaluated 13 secondary studies between 2012 and 2018. They consolidated the TD types found in previous secondary studies, identified a list of situations in which TD items can be found in software projects, and presented a map representing the activities, strategies, and tools supporting TDM. They pointed out TDM activities that do not yet have any support tool. According to Rios et al., [53], TD measurement has some kind of support in terms of tools. The different aspects of measurement discussed in their study are TD impact measurement, principal estimation, interest estimation, interest probability estimation, interest uncertainty estimation, and measurement in general. Most of these aspects are captured in our study, where we explored the concepts sufficient to capture TD quantification. Therefore,

researchers can now use our set of abstract TDQ concepts to analyze existing primary and secondary studies through the lens of TD Quantification.

Da Silva et al. [21] analyzed TDM tools. Their goal was to to consolidate the understanding about how existing TD tools map to different TD types and TDM activities, and to analyze the existing empirical evidence on the validity of the tools. The study found that most of the TD tools address code-related TD and that there are also some dedicated TDM tools for managing non-code-related TD. Our findings similarly suggest that the current focus on quantification approaches is on the code level. Therefore, we encourage the investigation of the quantification of other TD types.

6.1.3 Measurement (quantification) of TD. Khomyakov et al. [33] investigated existing tools for the measurement of TD, but they focused only on quantitative methods that could be automated. They reported on 38 papers out of 835 retrieved in their initial search. They found that almost all of the methods propose novel approaches to measure TD using different criteria. The authors claim that the research area of TD measurement is not mature and lacks independent evaluations of the methods proposed. Furthermore, they state that existing methods commonly focus on proposing new approaches, and therefore, no consolidation can be identified among the existing approaches. Contributions of our study help to fill this gap by proposing a consolidated set of abstract TDQ concepts that helps analyze existing quantification approaches by representing them via a common set of concepts.

Avgeriou et al. [9] compared a few existing tools measuring TD. They compared the features and the popularity of the tools. They focused on the TD types: code, design, and architecture. However, their study is limited to tools that estimate TD principal or interest. Our research complements their research by extending the discussion of TD quantification to a greater degree without limiting it to TD principal and interest. We discuss 22 concepts that capture TD quantification more holistically, covering the themes of process/time, cost, benefit, probability, and priority. Our study complements their study by making contributions towards filling the gaps discussed by Avgeriou et al. [9], one of them being the difficulties practitioners face trying to select a tool to match their needs and the second, the concept of TD and its role in software development being blurred. By classifying quantification approaches based on the set of abstract TDQ concepts and their high-level themes, we allow practitioners to use this classification to decide if a given quantification approach actually quantifies what they require to be quantified. The classification scheme is reusable. Hence, practitioners could use it to classify new approaches as well. Our findings emphasize the need to explore the quantification of TD by investigating how it could also be used strategically, for example, by quantifying the benefit of taking TD.

6.1.4 TD Prioritization. Alfayez et al. [2] investigated TD prioritization approaches and the prioritization techniques utilized by those approaches. Furthermore, they analyzed prioritization approaches based on their accounts for value, cost, or resource constraints. Leanarduzzi et al. [37] reviewed articles on technical debt prioritization, including strategies, processes, factors, and tools. They discovered that there is a lack of empirical evidence on measuring TD and that there is no validated, widely used set of tools specific to TD prioritization. Our study identifies the concepts sufficient to capture TD quantification. Prioritization strategies can benefit from this.

6.1.5 Decision Making in TD Management. Fernández-Sánchez et al. [26] identified elements required to manage TD. The elements were classified into three groups: basic decision-making factors, cost estimation techniques, practices, and techniques for decision-making. The factors were grouped based on stakeholders' points of view: engineering, engineering management, and business-organizational management. TD Item, Principal, Interest, and Interest probability were

among the elements identified by the authors, and these concepts have also been captured in our study.

Ribeiro et al. [52] introduce criteria that can be utilized for TDM decision-making. They evaluated the appropriate time for paying a TD item and how to apply decision-making criteria to balance the short-term benefits against long-term costs. They identified 14 decision-making criteria that development teams could use to prioritize the payment of TD items and a list of types of debt related to the criteria. They identify *'Debt impact on the project'* and *'Cost-Benefit'* as the most explored criteria in the studies captured in their mapping study. This confirms our selection of abstract TDQ concepts (i.e., concepts related to Cost and Benefit, including TD Interest) for discussing TD Quantification.

6.1.6 Financial aspect of TD. Ampatzoglou et al. [5] focused on the financial aspect of TD. The authors provided a glossary of financial terms and a classification scheme for financial approaches to managing TD. The authors motivate their study by stating that the measurement (or quantification) activity of TD will be supported by examining the financial perspective of technical debt. According to the authors, the most common financial terms that are used in technical debt research are principal and interest. The abstract TDQ concepts, TD Remediation Cost, and TD Interest identified in our study align with those terms. Repayment is another term in their glossary that also aligns with our concepts TD Remediation Step and TD Remediation Cost.

6.1.7 Architecture TD (ATD). Besker et al. [12] investigated Architecture TD (ATD) in their systematic literature review. They provided a comprehensive interpretation of the ATD phenomenon by contributing with a descriptive model categorizing the main characteristics of ATD.

6.1.8 TD in the context of Agile Software Development (ASD). Behutiye et al. [10] analyzed the state of the art of TD and its causes, consequences, and management strategies in the context of agile software development (ASD).

Table 4. Study Focus and Contributions of other secondary and tertiary studies along with Focus and Contributions of ours. | Italics — studies focusing on TD Measurement, Bold Italics — Our Study

Study	**Focus**	**Contribution**
Tom et al. [60]	Concept of TD	Theoretical framework comprising TD dimensions, attributes, precedents, outcomes
Alves et al. [4]	Concept of TD	Ontology of terms on TD Different types of TD Indicators
Alves et al. [3]	Concept of TD, TDM strategies	Ontology of terms on TD List of indicators to identify TD List of management strategies Data sources used in TD identification activities Software visualization techniques used to identify and manage TD
Li et al. [39]	TDM, TDM strategies and tools	10 types of TD 8 types of TDM activities 29 tools for TDM

Rios et al. [53]	TDM, TDM strategies and tools	Consolidated list of TD types List of situations in which TD items can be found in software projects Map representing activities, strategies, and tools supporting TDM TDM activities that do not yet have a support tool
Da Silva et al. [21]	TDM tools	Consolidated the understanding of how existing TD tools map to different TD types and TDM activities Analyzed the existing empirical evidence on the validity of the tools.
Khomyakov et al. [33]	TD measurement	Consolidated (automated) tools for the measurement of TD
Avegeriou et al. [8]	TD measurement	Compared existing tools to measure Code, Design, and Architecture TD but focused on the popularity and features of the tools
Our Study	TD measurement (quantification)	A set of abstract TDQ concepts sufficient to capture TD quantification conceptually. This can serve as a reference to develop new quantification approaches. A classification scheme based on the set of abstract TD quantification concepts and their high-level themes process/time, cost, benefit, probability, and priority. The classification scheme helped understand gaps in the literature to provide future research directions for researchers. Practitioners could also make use of the classification to select existing quantification approaches fit for their particular needs.
Alfayaz et al. [2]	TD Prioritization	Prioritized approaches based on their value, cost and resources
Lenarduzzi et al. [37]	Prioritization	Strategies, processes, factors and tools for prioritization
Fernandez-Sanches et al. [25]	Decision-making	Elements required to manage TD classified into basic decision-making factors, cost estimation techniques, practices, and techniques for decision-making — factors were grouped based on stakeholder's point of view: engineering, engineering management and business organization management
Ribeiro et al. [52]	Decision-making	14 decision-making criteria that development teams can use to prioritize the payment of TD items List of types of debt related to the criteria
Ampatzoglou et al. [6]	Financial aspect of TD	Glossary of terms Classification Scheme for financial approaches to manage TD

Besker et al. [12]	Architecture TD	Descriptive model categorizing the main characteristics of ATD
Behutiye et al. [10]	TD in the context of ASD	Analyzed the state of the art of TD and its causes, consequences, and management strategies in the context of ASD

6.2 TD, TDM and Quantification of TD

The first mention of TD as a metaphor to indicate writing not quite-right code as a trade-off of long-term code quality for a short-term gain, was by Cunningham in 1992 [19]. Avgeriou et al. proposed a consensus definition for TD at a Dagstuhl seminar held in 2016, referred to as the 16162 definition of TD: *"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability [8]."*

Avgeriou et al. [8] also introduced a conceptual model for TD (referred to as the '16162 model' in this paper) based on two viewpoints. The *first viewpoint* describes the properties, artifacts, and elements related to TD items, and the *second viewpoint* articulates the management and process-related activities or the different states that debt may go through. In the 16162 model, TD was described as one of many concerns in a software system. The authors also discussed the concept of a TD item. A TD item is associated with one or more artifacts of the software development process, such as code, test, or documentation, and is caused by, for example, schedule pressure. The concept of a TD Item was captured in our work as one of the predominant abstract TDQ concepts discussed and quantified in the literature.

However, the 16162 model does not entirely capture the two viewpoints they initially discuss. Instead, it focuses more on the first viewpoint, capturing the elements related to TD. Therefore, the model does not discuss the quantification (or measurement) of TD, which is one of the TD Management (TDM) activities introduced by Li et al. [39] in a previous study. According to Li et al., TDM includes activities that prevent potential TD from being incurred (e.g., prevention) and activities that deal with accumulated TD to make it visible, controllable and to keep a balance between costs and value of a software project (e.g., identification, visualizing, monitoring, measurement, prioritization, repayment). In our work, we focus on TD quantification (measurement), as one of the main TDM activities which can support making informed decisions for TDM.

Our work establishes the first theoretical foundation for understanding the quantification of TD by consolidating concepts sufficient to capture the phenomenon of quantification of TD. Furthermore, we categorize these concepts into high-level themes of process/time, cost, benefit, probability, and priority and show which concepts are represented in the existing literature and which concepts are supported by metrics in the existing literature. The classification scheme developed based on the consolidated set of abstract TDQ concepts and their high-level themes can serve as a reusable tool to analyze new quantification approaches. The set of abstract TDQ concepts can also serve as a reference to develop new quantification approaches.

7 CONCLUSION

We conducted a systematic mapping study to investigate what approaches to TD quantification have been proposed in the research literature for Code, Design, and Architecture TD. We found 39 unique TD quantification approaches. We analyzed what TD quantification concepts are discussed in these quantification approaches and which of those concepts are supported by metrics based

on a classification scheme we developed during our study. The classification scheme comprises a set of abstract TDQ concepts and their high-level themes, *process/time, cost, benefit, probability,* and *priority*. The classification of TD quantification approaches based on the classification scheme helped identify gaps in the research literature and provide recommendations for researchers and practitioners.

Among the TD quantification concepts discussed in the different quantification approaches, *TD Item, TD remediation cost, TD Interest, and Benefit of remediating TD* were the most frequently discussed abstract TDQ concepts. These concepts were also supported by metrics for their quantification. Some abstract TDQ concepts were poorly examined, for example, the *benefit of taking TD*. It was evident from our findings that cost concepts were more frequently discussed and supported by metrics in the quantification approaches, while benefit concepts were not. Furthermore, the focus was on remediating TD in retrospect rather than using TD strategically as intended by the metaphor. This raises the question of whether the existing quantification approaches proposed in the research literature quantify TD in a reliable manner that supports TDM decision-making.

Our work serves as a theoretical foundation to understand and analyze TD quantification approaches conceptually. The consolidated set of abstract TDQ concepts we developed is a uniform representation of what is discussed in terms of TD quantification in the literature. Therefore, the set of abstract TDQ concepts and their high-level themes can serve as a reference point to develop new quantification approaches. In future work, we plan to identify the relationships between the different abstract TDQ concepts to develop a conceptual model to shed light on how quantifying these concepts could better support TDM decision-making.

REFERENCES

[1] Md Abdullah Al Mamun, Antonio Martini, Miroslaw Staron, Christian Berger, and Jörgen Hansson. 2019. Evolution of technical debt: An exploratory study. In *2019 Joint Conference of the International Workshop on Software Measurement and the International Conference on Software Process and Product Measurement, IWSM-Mensura 2019, Haarlem, The Netherlands, October 7-9, 2019*, Vol. 2476. CEUR-WS, 87–102.

[2] 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.

[3] 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.

[4] Nicolli SR Alves, Leilane F Ribeiro, Vivyane Caires, Thiago S Mendes, and Rodrigo O Spínola. 2014. Towards an ontology of terms on technical debt. In *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 1–7.

[5] Areti Ampatzoglou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2015. The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology* 64 (2015), 52–73.

[6] Areti Ampatzoglou, Alexandros Michailidis, Christos Sarikyriakidis, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2018. A framework for managing interest in technical debt: an industrial validation. In *Proceedings of the 2018 International Conference on Technical Debt*. 115–124.

[7] Arooj Arif and Zeeshan Ali Rana. 2020. Refactoring of code to remove technical debt and reduce maintenance effort. In *2020 14th International Conference on Open Source Systems and Technologies (ICOSST)*. IEEE, 1–7.

[8] Paris Avgeriou, Philippe Kruchten, Ipek Ozkaya, and Carolyn Seaman. 2016. Managing technical debt in software engineering (dagstuhl seminar 16162). In *Dagstuhl Reports*, Vol. 6. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[9] 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.

[10] Woubshet Nema Behutiye, Pilar Rodríguez, Markku Oivo, and Ayşe Tosun. 2017. Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. *Information and Software Technology* 82 (2017), 139–158.

[11] Terese Besker, Antonio Martini, and Jan Bosch. 2017. The pricey bill of technical debt: When and by whom will it be paid?. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 13–23.

[12] Terese Besker, Antonio Martini, and Jan Bosch. 2018. Managing architectural technical debt: A unified model and systematic literature review. *Journal of Systems and Software* 135 (2018), 1–16.
[13] Terese Besker, Antonio Martini, and Jan Bosch. 2019. Technical Debt Triage in Backlog Management. In *2019 IEEE/ACM International Conference on Technical Debt (TechDebt)*. 13–22. https://doi.org/10.1109/TechDebt.2019.00010
[14] Pearl Brereton, Barbara A Kitchenham, David Budgen, Mark Turner, and Mohamed Khalil. 2007. Lessons from applying the systematic literature review process within the software engineering domain. *Journal of systems and software* 80, 4 (2007), 571–583.
[15] Antonio Cavacini. 2015. What is the best database for computer science journal articles? *Scientometrics* 102, 3 (2015), 2059–2071.
[16] Sofia Charalampidou, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Paris Avgeriou. 2017. Assessing code smell interest probability: a case study. In *Proceedings of the XP2017 Scientific Workshops*. 1–8.
[17] Aabha Choudhary and Paramvir Singh. 2016. Minimizing Refactoring Effort through Prioritization of Classes based on Historical, Architectural and Code Smell Information.. In *QuASoQ/TDA@ APSEC*. 76–79.
[18] Victoria Clarke and Virginia Braun. 2017. Thematic analysis. *The journal of positive psychology* 12, 3 (2017), 297–298.
[19] Ward Cunningham. 1992. The WyCash portfolio management system. *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA* Part F1296, October (1992), 29–30. https://doi.org/10.1145/157709.157715
[20] Bill Curtis, Jay Sappidi, and Alexandra Szynkarski. 2012. Estimating the size, cost, and types of technical debt. In *2012 Third International Workshop on Managing Technical Debt (MTD)*. IEEE, 49–53.
[21] José Diego Saraiva da Silva, José Gameleira Neto, Uirá Kulesza, Guilherme Freitas, Rodrigo Reboucas, and Roberta Coelho. 2021. Exploring Technical Debt Tools: A Systematic Mapping Study. In *International Conference on Enterprise Information Systems*. Springer, 280–303.
[22] Georgios Digkas, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Paris Avgeriou, Oliviu Matei, and Robert Heb. 2021. The risk of generating technical debt interest: a case study. *SN Computer Science* 2, 1 (2021), 1–12.
[23] Robert J Eisenberg. 2012. A threshold based approach to technical debt. *ACM SIGSOFT Software Engineering Notes* 37, 2 (2012), 1–6.
[24] Davide Falessi and Andreas Reichel. 2015. Towards an open-source tool for measuring and visualizing the interest of technical debt. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 1–8.
[25] Carlos Fernández-Sánchez, Jessica Díaz, Jennifer Pérez, and Juan Garbajosa. 2014. Guiding flexibility investment in agile architecting. In *2014 47th Hawaii International Conference on System Sciences*. IEEE, 4807–4816.
[26] Carlos Fernández-Sánchez, Juan Garbajosa, Agustín Yagüe, and Jennifer Perez. 2017. Identification and analysis of the elements required to manage technical debt by means of a systematic mapping study. *Journal of Systems and Software* 124 (2017), 22–38.
[27] Francesca Arcelli Fontana, Vincenzo Ferme, Marco Zanoni, and Riccardo Roveda. 2015. Towards a prioritization of code debt: A code smell intensity index. In *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*. IEEE, 16–24.
[28] Martin Fowler. 2019. Is high quality software worth the cost? https://martinfowler.com/articles/is-quality-worth-cost.html
[29] Yuepu Guo and Carolyn Seaman. 2011. A portfolio approach to technical debt management. In *Proceedings of the 2nd Workshop on Managing Technical Debt*. 31–34.
[30] Yuepu Guo, Carolyn Seaman, Rebeka Gomes, Antonio Cavalcanti, Graziela Tonin, Fabio Q. B. Da Silva, Andre L. M. Santos, and Clauirton Siebra. 2011. Tracking Technical Debt – An Exploratory Case Study. In *Proceedings of the 2011 27th IEEE International Conference on Software Maintenance (ICSM '11)*. IEEE Computer Society, USA, 528–531. https://doi.org/10.1109/ICSM.2011.6080824
[31] Trong Tan Ho and Guenther Ruhe. 2014. When-to-release decisions in consideration of technical debt. In *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 31–34.
[32] Rick Kazman, Yuanfang Cai, Ran Mo, Qiong Feng, Lu Xiao, Serge Haziyev, Volodymyr Fedak, and Andriy Shapochka. 2015. A Case Study in Locating the Architectural Roots of Technical Debt. *Proceedings - International Conference on Software Engineering* 2 (2015), 179–188. https://doi.org/10.1109/ICSE.2015.146
[33] Ilya Khomyakov, Zufar Makhmutov, Ruzilya Mirgalimova, and Alberto Sillitti. 2020. An analysis of automated technical debt measurement. In *Enterprise Information Systems: 21st International Conference, ICEIS 2019, Heraklion, Crete, Greece, May 3–5, 2019, Revised Selected Papers 21*. Springer, 250–273.
[34] Barbara Kitchenham, Rialette Pretorius, David Budgen, O Pearl Brereton, Mark Turner, Mahmood Niazi, and Stephen Linkman. 2010. Systematic literature reviews in software engineering–a tertiary study. *Information and software technology* 52, 8 (2010), 792–805.
[35] Boris Kontsevoi, Denis Syraeshko, and Sergei Terekhov. 2022. Practice of Tech Debt Assessment and Management with TETRA™. In *Proceedings of Sixth International Congress on Information and Communication Technology*. Springer,

843–850.

[36] Makrina Viola Kosti, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, Georgios Pallas, Ioannis Stamelos, and Lefteris Angelis. 2017. Technical debt principal assessment through structural metrics. In *2017 43rd Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 329–333.

[37] 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.

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

[39] 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. https://doi.org/10.1016/j.jss.2014.12.027

[40] Antonio Martini. 2018. Anacondebt. (2018), 55–56. https://doi.org/10.1145/3194164.3194185

[41] Antonio Martini, Jan Bosch, and Michel Chaudron. 2015. Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study. *Information and Software Technology* 67 (2015), 237–253. https://doi.org/10.1016/j.infsof.2015.07.005

[42] Antonio Martini, Erik Sikander, and Niel Madlani. 2018. A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component. *Information and Software Technology* 93 (2018), 264–279.

[43] Alois Mayr, Reinhold Plösch, and Christian Körner. 2014. A benchmarking-based model for technical debt calculation. In *2014 14th International Conference on Quality Software*. IEEE, 305–314.

[44] Solomon Mensah, Jacky Keung, Michael Franklin Bosu, and Kwabena Ebo Bennin. 2016. Rework effort estimation of self-admitted technical debt. (2016).

[45] Nikolaos Nikolaidis, Dimitrios Zisis, Apostolos Ampatzoglou, Alexander Chatzigeorgiou, and Dimitrios Soudris. 2021. Experience With Managing Technical Debt in Scientific Software Development Using the EXA2PRO Framework. *IEEE Access* 9 (2021), 72524–72534.

[46] Robert L. Nord, Ipek Ozkaya, Philippe Kruchten, and Marco Gonzalez-Rojas. 2012. In search of a metric for managing architectural technical debt. *Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012* (2012), 91–100. https://doi.org/10.1109/WICSA-ECSA.212.17

[47] Ariadi Nugroho, Joost Visser, and Tobias Kuipers. 2011. An empirical model of technical debt and interest. In *Proceedings of the 2nd workshop on managing technical debt*. 1–8.

[48] 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.

[49] Kai Petersen, Robert Feldt, Shahid Mujtaba, and Michael Mattsson. 2008. Systematic mapping studies in software engineering. In *12th International Conference on Evaluation and Assessment in Software Engineering (EASE) 12*. 1–10.

[50] Kai Petersen, Sairam Vakkalanka, and Ludwik Kuzniarz. 2015. Guidelines for conducting systematic mapping studies in software engineering: An update. *Information and software technology* 64 (2015), 1–18.

[51] Derek Reimanis and Clemente Izurieta. 2016. Towards Assessing the Technical Debt of Undesired Software Behaviors in Design Patterns. *Proceedings - 2016 IEEE 8th International Workshop on Managing Technical Debt, MTD 2016* (2016), 24–27. https://doi.org/10.1109/MTD.2016.13

[52] Leilane Ferreira Ribeiro, Mário André de Freitas Farias, Manoel G Mendonça, and Rodrigo Oliveira Spínola. 2016. Decision Criteria for the Payment of Technical Debt in Software Projects: A Systematic Mapping Study.. In *ICEIS (1)*. 572–579.

[53] 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, February (2018), 117–145. https://doi.org/10.1016/j.infsof.2018.05.010

[54] Riccardo Roveda, Francesca Arcelli Fontana, Ilaria Pigazzini, and Marco Zanoni. 2018. Towards an architectural debt index. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 408–416.

[55] Klaus Schmid. 2013. A Formal Approach to Technical Debt Decision Making. (2013), 153–162.

[56] Andriy Shapochka and Borys Omelayenko. 2016. Practical Technical Debt Discovery by Matching Patterns in Assessment Graph. In *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*. IEEE, 32–35.

[57] Tushar Sharma, Pratibha Mishra, and Rohit Tiwari. 2016. Designite - A software design quality assessment tool. *Proceedings - 1st International Workshop on Bringing Architectural Design Thinking Into Developers' Daily Activities, Bridge 2016* (2016), 1–4. https://doi.org/10.1145/2896935.2896938

[58] Vallary Singh, Will Snipes, and Nicholas A Kraft. 2014. A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. In *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 27–30.

[59] Marek G Stochel, Piotr Chołda, and Mariusz R Wawrowski. 2020. Continuous debt valuation approach (codva) for technical debt prioritization. In *2020 46th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 362–366.

[60] Edith Tom, Aybüke Aurum, and Richard Vidgen. 2013. An exploration of technical debt. *Journal of Systems and Software* 86, 6 (2013), 1498–1516.

[61] Adam Tornhill. 2018. Assessing technical debt in automated tests with codescene. In *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. IEEE, 122–125.

[62] Sri Harsha Vathsavayi and Kari Systä. 2016. Technical debt management with genetic algorithms. In *2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 50–53.

[63] Urjaswala Vora. 2022. Measuring the Technical Debt. In *2022 17th Annual System of Systems Engineering Conference (SOSE)*. IEEE, 185–189.

[64] Claes Wohlin. 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In *Proceedings of the 18th international conference on evaluation and assessment in software engineering*. 1–10.

[65] Nico Zazworka, Carolyn Seaman, and Forrest Shull. 2011. Prioritizing design debt investment opportunities. In *Proceedings of the 2nd Workshop on Managing Technical Debt*. 39–42.

A QUANTIFICATION APPROACHES

Table 5. Quantification Approaches found in the SMS for code-related types of TD | QAn — Quantification Approach

Quant. App.	Title of the Article	Citation	Pub Year	TD Type
QA1	Assessing Technical Debt in Automated Tests with CodeScene	[61]	2018	Code
QA2	Designite - A Software Design Quality Assessment Tool	[57]	2016	Design
QA3	AnaconDebt: A Tool to Assess and Track Technical Debt	[40]	2018	Architecture
QA4	A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension	[58]	2014	Code
QA5	Towards assessing the Technical Debt of Undesired Software Behaviors in Design Patterns	[51]	2016	Design
QA6	Technical Debt Principal Assessment through Structural Metrics	[36]	2017	Code
QA7	Rework Effort Estimation of Self-admitted Technical Debt	[44]	2016	Design
QA8	Technical Debt Management with Genetic Algorithms	[62]	2016	Architecture
QA9	Towards an Architectural Debt Index	[54]	2018	Architecture
QA10	Estimating the Size, Cost, and Types of Technical Debt	[20]	2012	Code and Architecture
QA11	A Framework for Managing Interest in Technical Debt: An Industrial Validation	[6]	2018	Code
QA12	A Proposed Model-Driven Approach to Manage Architectural Technical Debt Life Cycle	[48]	2019	General
QA13	Practical Technical Debt Discovery by Matching Patterns in Assessment Graph	[56]	2016	Architecture
QA14	Minimizing Refactoring Effort through Prioritization of Classes based on Historical, Architectural and Code Smell Information	[17]	2016	Code and Architecture
QA15	Prioritizing Design Debt Investment Opportunities	[65]	2011	Design
QA16	Assessing Code Smell Interest Probability: A Case Study	[16]	2017	Code
QA17	Towards a prioritization of code debt: A code smell Intensity Index	[27]	2015	Code
QA18	Managing Technical Debt with the SQALE Method	[38]	2012	General
QA19	A portfolio approach to technical debt management	[29]	2011	General

QA20	A Formal Approach to Technical Debt Decision Making	[55]	2013	Architecture
QA21	A Benchmarking-based model for Technical Debt Calculation	[43]	2014	General
QA22	A proposed sizing model for managing 3rd party Code Technical Debt	[48]	2018	General
QA23	When-to-release decisions in consideration of TD	[31]	2014	General
QA24	Tracking technical debt - An exploratory case study	[30]	2011	General
QA25	Guilding Flexibility Investment in Agile Architecting	[25]	2014	Architecture
QA26	A Threshold based approach to TD	[23]	2012	General
QA27	Estimating the Principal of an Application's Technical Debt	[20]	2012	General
QA28	A semi-automated framework for the identification and estimation of Architectural Technical Debt: A comparative case-study on the modularization of a software component	[42]	2018	Architecture
QA29	Evolution of TD: An exploration Study	[1]	2019	General
QA30	An Empirical Model of Technical Debt and Interest (SIG method)	[47]	2011	General
QA31	In Search of a Metric for Managing Architectural Technical Debt	[46]	2012	Architecture
QA32	A Case Study in Locating the Architectural Roots of Technical Debt	[32]	2015	Architecture
QA33	Towards an open-source tool for measuring and visualizing the interest of technical debt	[24]	2015	General
QA34	Measuring the Technical Debt	[63]	2022	Code
QA35	Practice of Tech Debt Assessment and Management with TETRA™	[35]	2022	Code
QA36	The Risk of Generating Technical Debt Interest: A Case Study	[22]	2021	Code
QA37	Refactoring of Code to Remove Technical Debt and Reduce Maintenance Effort	[7]	2020	Code
QA38	Continuous Debt Valuation Approach (CoDVA) for Technical Debt Prioritization	[59]	2020	General
QA39	Experience With Managing Technical Debt in Scientific Software Development Using the EXA2PRO Framework	[45]	2021	Design

B APPROACH CONCEPTS MAPPED TO ABSTRACT TDQ CONCEPTS — MAPPING D

Table 6. Approach Concepts mapped to Abstract TDQ Concepts (Mapping D) | Bold Italics — highest counts

Abstract TDQ Concept	**Num. of QAs**	**Num. of Approach Concepts mapped from QAs**	**Concepts mapped from QAs**
Product	3	3	QA23: Product, QA31: Product, QA38: Product
Release	3	3	QA23: Release, QA31: Release, QA38: Release
Dev Path	3	3	QA20: Evolution Sequence, QA31:Dev Path, QA38: Feature Pipeline or Roadmap
Dev Step	2	2	QA20: Evolution Step, QA34: Development phase
Impl Step	0	0	-
Remediation Step	6	6	QA14: Pay off TD (through refactoring), QA15: Refactoring, QA32:Refactoring Step, QA35: Elimination Step, QA36: TD remediation, QA37: Refactoring
Feature	4	4	QA8: Features, QA23: Feature, QA31:Feature, QA38: Feature
TD Item	***19***	***19***	QA1:Refactoring Candidates, QA2:Design smells, QA3: TD Item, QA8: Technical Debt Items, QA12:ATD Item, QA13:TD Item, QA15:God Classes, QA17:Code Smell, QA18:Debt Item, QA19:TD Item, QA21: violations to be fixed, QA24:TD Items, QA27:Should-fix Violations, QA28:non-modularized components (lack of modularization), QA29:Code smells, QA32:Architectural Flaw, QA37: Code smell, QA38: TD Item, QA39: TD Item or design problem
Dev Path Total Cost	2	3	QA20: Evolution sequence cost (Change cost of evol. sequence), QA31:Cumulative Total Cost, QA31:Percentage of Cumulative Total Cost
Dev Step Total Cost	2	3	QA20: Evolution Step Cost or (Change Cost), QA31:Cumulative Total Cost, QA31:Percentage of Cumulative Total Cost
Impl Step Total Cost	2	3	QA23: Effort required to Implement a Feature, QA31:Cumulative Total Cost, QA31:Percentage of Cumulative Total Cost

Remediation Cost	***24***	***25***	QA3: Cost of refactoring (Principal), QA6: TD Principal, QA8: Cost to fully eliminate the debt (Principal), QA10:Principal (Cost to fix the problems), QA11:Principal, QA13:Refactoring cost, QA15:Cost of refactoring (overall metric score), QA18:Time to remediate each debt item or remediation cost. (Principal), QA18:technical debt, QA19:Principal, QA20: Cost associated with refactoring, QA21: Remediation Cost, QA24:Principal, QA25:Cost of eliminating the weaknesses (Principal), QA26: Debt remediation effort, QA27:Principal (Cost of remediatnig should-fix violations), QA28:Principal (Cost of reworking or refactoring the component), QA29:Principal of code smell (time to fix it), QA30:Repair Effort (RE) or cost of repair (cost to repair quality issues to reach the ideal quality level), QA32:Cost of Refactoring, QA34: Principal or Refactoring Cost, QA34: TD Principal or efforts required to refactor, QA37: Man-hour used to remove the selected code smell, QA38: Principal or cost of refactoring, QA39: Principal or time required to fix or cost to resolve each problem
Impl Cost of Feature	4	4	QA8: Implementation cost, QA31:Implementation Cost, QA37: Effort required to add features or Man-hour for Addition of Functionality, QA38: Development cost of Feature
TD Interest	***23***	***24***	QA1:Impact on maintenance effort, QA3: Interest (extra costs), QA4: Difference between code comprehension effort in ideal and current state Interest (gap between maintenance costs under ideal conditions versus conditions where maintenance is higher due to accrued debt), QA10:Interest, QA11:Interest, QA15:Impact of god class on quality attributes (defect likelyhod, change likelihood), QA18:Impact of the debt items on the business or non-remediation cost (Interest), QA18:business impact, QA19:Expected Interest Amount, QA20: Additional development cost introduced in evolution, QA21: Non Remediation Cost, QA24:Interest Amount, QA25: Additional Cost of Change (CoC) - Additional cost derived from lack of flexibility or additional cost of implementing changes (Interest), QA27:Interest (Continuing cost of not remediating should-fix violations), QA28:Interest (code somplexity, maintenance costs), QA30:Extra maintenance cost spent for not achieving the ideal quality level (diff between maintenance cost of current level and ideal level), QA32:Penalty Incurred by Debts, QA34: Interest, QA34: Interest or Impact of technical debt, QA35: TD Impact, QA36: TD Interest, QA38: Interest or Impact, QA39: Interest

New Code Cost not associated with TD	0	0	-
Rework Cost not associated with TD	0	0	-
New Code Cost associated with TD	0	0	-
Rework Cost associated with TD	2	2	QA8: Extra cost to selected new features due to unfixed technical debt, QA31:Rework Cost
Benefit of Remediating TD	***9***	***11***	P9:Architectural Debt Index (ADI), QA20: Refactoring benefit, QA25:ROI of designing for flexibility, QA28:benefit of refactoring a non modularized component, QA30:ROI, QA30:NPV, QA32:Expected benefit of Refactoring, QA37: Impact of removing TD, QA38: Return on Investment (ROI), QA38: Benefits from paying off known technical debt items or effort saved (monetized value or benefit associated with any of refactorings), QA39: Benefit
Benefit of Taking TD	0	0	-
Interest Probability	4	4	QA16:Smell Interest Probability, QA24:Interest Probability, QA36: Interest Probability, QA39: Interest Probability
Priority	5	5	QA1:Priority, QA3: Priority, QA13:Priority, QA15:Priority of god class according to cost benefit analysis (impact vs effort or cost of refactoring), QA17:Priority

C APPROACH CONCEPTS MAPPED TO ABSTRACT TDQ CONCEPTS — MAPPING I

Table 7. Approach Concepts mapped to Abstract TDQ Concepts (Mapping I) | Bold Italics — highest counts

Abstract TDQ Concept	**Num. of QAs**	**Num. of Approach Concepts mapped from QAs**	**Concepts mapped from QAs**
Product	1	1	QA32:Product
Release	1	1	QA32:Release
Dev Path	1	1	QA32:Dev Path
Dev Step	2	2	QA31:Dev Step, QA32:Dev Step
Impl Step	2	2	QA31:Implementation Step, QA32:Implementation Step
Refactoring Step	4	7	QA11: Number of problems that must be fixed, QA11:time required to fix each problem, QA11:cost for fixing each problem, QA22: migrate the API calls in the using code to the newer version of the API, QA25: Eliminating quality weaknesses, QA32:Cost of Refactoring, QA39: Refactoring or repayment
Feature	4	5	QA4: Change Task, QA8: Business value of feature, QA8: Short term value (Benefit of focusing on features), QA32:Feature, QA38: Sales opportunity
TD Item	***9***	***11***	QA9:Architectural Smells, QA10:Must fix problems, QA11:Problems that must be fixed, QA12:Architectural Smells, QA12:Architectural Anti-Patterns, QA12:Sub-optimal architectural decisions, QA14:Code smells, QA16:Code Smell, QA23: Technical Debt, QA27:Technical Debt (Future costs attributes to known violations in production code that should be fixed, includes both Principal and Interest), QA35: Critical Problem
Dev Path Total Cost	4	5	QA4: Developer code comprehension effort, QA8: Total implementation cost of a sprint, QA23:Total Release Value (sum of all values of individual features offered in that release), QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed
Dev Step Total Cost	3	4	QA4: Developer code comprehension effort, QA8: Total implementation cost of a sprint, QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed
Impl Step Total Cost	2	3	QA8: Total implementation cost of a sprint, QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed

Remediation Cost	6	6	QA1:Rework, QA7:Rework Effort, QA14:Estimated effort of refactoring (the total number of classes that needs to be refactored divided by the total number of classes present in the software), QA22:Principal cost (Cost to migrate the API calls in the using code to the newer version of the API), QA23:Effort required to pay off accumulated TD in the current release, QA26:Cost to return to Green
Impl Cost of Feat	1	2	QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed
TD Interest	6	6	QA2:Smell density, QA8: Extra effort, QA12:ATD Item Impact, QA22:Accumulated Principal Cost (Cost of differing upgrades), QA31:Rework Cost, QA36: New Code
New Code Cost not associated with TD	3	4	QA31:Implementation Cost, QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed, QA36: New Code
Rework Cost not associated with TD	1	2	QA32:Current Total LOC Changed, QA32:Expected Total LOC Changed
New Code Cost associated with TD	***12***	***12***	QA1:Impact on maintenance effort, QA2:Smell density, QA3: Interest (extra costs), QA4: Difference between code comprehension effort in ideal and current state, QA5: Program comprehension effort (derived), QA10:Interest, QA11:Interest, QA12:ATD Item Impact, QA15:Impact of god class on quality attributes (defect likelyhod, change likelihood), QA18:Impact of the debt items on the business or non-remediation cost (Interest), QA32:Penalty Incurred by Debts, QA36: New Code
Rework Cost associated with TD	***13***	***13***	QA1:Impact on maintenance effort, QA2:Smell density, QA3: Interest (extra costs), QA4: Difference between code comprehension effort in ideal and current state, QA5: Program comprehension effort (derived), QA8: Future extra costs of debt item, QA10:Interest, QA11:Interest, QA12:ATD Item Impact, QA15:Impact of god class on quality attributes (defect likelyhod, change likelihood), QA18:Impact of the debt items on the business or non-remediation cost (Interest), QA24:Effort to rewrite, QA32:Penalty Incurred by Debts
Benefit of Remediating TD	7	8	QA1: Effect of a refactoring, QA8: Future investment value (Benefit of paying the debt items), QA14:Code smells correction ratio (the total number of code smell instances to be removed by refactoring the prioritized classes, divided by the total number of code smell instances present in the software), QA14: total reduction in refactoring effort, QA18:SQALE debt ratio (technical debt divided by the budget of the project), QA19:Net benefit (Principal - Interest amount), QA23: Potential effort the team can invest in the next release, QA34: ROI of Refactoring

Benefit of taking TD	3	3	QA8: Short term value (Benefit of focusing on features), QA18:SQALE debt ratio (technical debt divided by the budget of the project), QA23: Potential effort the team can invest in the next release
Interest Probability	5	6	QA8: Dependency of a feature on a set of debt items, QA9:Page Rank (estimates whther the AS is located in an important part of the project), QA19:Interest Standard Deviation or variance of return or risk that a TD item will not produce benefit, QA19:Correlations with other debt items, QA20: Probability measure (or uncertainty of evol. steps), QA25:Uncertainity that a change could happen
Priority	7	7	QA9:Severity of an Architectural Smell, QA10:Potential severity (high , medium, low), QA12:Severity, QA14:Rank (According to class score), QA18:SQALE rating (a grade a, B, C assigned based on low or high debt ratio), QA26:Priority of metrics, QA28:Refactor index

D APPROACH CONCEPTS MAPPED TO ABSTRACT TDQ CONCEPTS — MAPPING M

Table 8. Metrics supporting the Quantification of Abstract TDQ Concepts | Italics — Concepts where one or more Quantification Approaches discussed Metrics, Bold Italics — highest counts

Abstract TDQ Concept	**Num. of QAs providing some form of measurement for concepts**	**Num. of Metrics mapped from QAs**	**Metrics mapped from QAs**
Product	0	0	-
Release	0	0	-
Dev Path	0	0	-
Dev Step	0	0	-
Impl Step	0	0	-
Refactoring Step	0	0	-
Feature	0	0	-
TD Item	*6*	*10*	QA9:Dependency metrics (Martin's), QA10:Total Quality Index, QA10:Halth factors e.g. robustness, QA13:Size of TD Item (LOC), QA13:Complexity of TD Item, QA23:maximum num of effort days the release could be varied, QA23:change in the value due to functionality change, QA23:remaining TD in that release, QA29:TD Density (amount of TD per 100 LOC), QA29:TD Density Trend (Slope of the line of two successive TD density measures)
Dev Path Total Cost	*2*	*5*	QA4: Time spent in class, QA4: Time spent in other classes, QA23:maximum num of effort days the release could be varied, QA23:change in the value due to functionality change, QA23:remaining TD in that release
Dev Step Total Cost	*1*	*2*	QA4: Time spent in class, QA4: Time spent in other classes
Impl Step Total Cost	0	0	-

Remediation Cost	***14***	***43***	QA6: Coupling metrics, QA6: Cohesion metrics, QA6: Inheritance metrics, QA6: Size metrics, QA6: Polymorephysm metrics, QA7:Commented LOC on average per SATD prone source file, QA10:Number of must fix problems, QA10:Potential severity (high , medium, low), QA10:Time required to fix each problem, QA10:Cost for fixing each problem(labor rate measured in dollars), QA11:Number of problems that must be fixed, QA11:time required to fix each problem, QA11:cost for fixing each problem, QA15: Refactoring, QA13:Refactoring size, QA13:Work rate, QA13:Fixed expenses, QA14:Number of classes to be refactored, QA14:Total number of classes in the system, QA15:WMC, QA15:TCC, QA15:AFTD, QA18:TD Index, QA21: target quality levels (defines the threshold for minimum quality), QA21: max allowed violations per 3-tuple, QA21: num of violations to be fixed (number of violations actually need to be fixed to reach the specified quality level for each 3-tuple), QA22:Size of the impact(LOC) upgrading an aging third-party component to the current version, QA22:Size of the code base using the thirs-party component, QA22:Degree of API deprecation between the version used and the current version of the third-party component, QA22:Age in years of the third-party software component version being used, QA23:TD adjustment factor (weighted average of all criteria in all 4 TD dimensions: process rules compliance, qualty testing, maintainability, complexity), QA23:Effort adjustment factor (product or project specific cost drivers that will affect the efficiency addressing TD e.g. availability of human experts or developers), QA26:Acceptable levels of debt, QA26:Debt per KSLOC, QA27:the number of should fix violations, QA27: percentage of violations to be fixed, QA27:hours to fix each violation, QA27:cost of labour, QA30:Rework Fraction (estimate of the percentage of lines of code that need to be changed to improve the quality of software to a higher level), QA30:Rebuild Value (estimate of effort in man-months), QA30:Refcatoring adjustment (context specific adjustment), QA30:Overall maintainability rating, QA30:Risk Profile
Impl Cost of Feat	0	0	-

Interest (c)	***12***	***37***	QA1:Complexity growth, QA1:Code growth, QA1:Comments growth, QA1:Change Frequency, QA1:Lines of Code, QA1:Cyclomatic Complexity, QA3: External propagation factors (e.g. num of TD related increments planned, num of users affected, QA3: Internal propagation factors (e.g. growth of source, growtyh of complexity), QA5: Excessive actions, QA5: Improper order of sequences, QA5: Program comprehension effort (derived), QA11:TD Breaking Point (in num of versions), QA12:Granularity, QA12:Severity, QA12:Amount of architectural decisions impacted by the ATD Business impact index, QA30:Rebuild Value (estimate of effort in man-months), QA30:Maintenance Fraction, QA30:Maintenance Effort, QA30:Quality Factor, QA34: Defect Proneness, QA34: Maximum Defects per 100 LOC touched, QA34: Extra Defect Proneness, QA34: Maximum Extra Defects per 100 LOC Touched, QA34: Relative Extra Defect Proneness, QA34: Average Relative Extra Defect Proneness, QA34: Violation Density, QA34: Linkage, QA34: Estimation Error, QA34: Complexity Index, QA34: Modularity Index, QA34: Data Coupling Index, QA34: Efforts Deviation Index, QA35: Metric Index, QA35: Dimension Index, QA35: TETRA™ Index, QA36: Interest Generation Risk Importance (IGRI)
New Code Cost not associated with TD	0	0	-
Rework Cost not associated with TD	0	0	-
New Code Cost associated with TD	0	0	-
Rework Cost associated with TD	*1*	*2*	QA31:Number of Dependencies, QA31:Change Propagation Metric
Benefit of Remediating TD	*2*	*7*	QA9:Severity of an Architectural Smell, QA9:Page Rank (estimates whther the AS is located in an important part of the project), QA9:AS weight (the number of dependencies associated with an AS), QA9:History score(score associated to the trend evolution), QA9:Architectural Smell Impact Score (ASIS), QA14:Number of code smells to be removed, QA14:Total number of code smells in the system
Benefit of taking TD	0	0	-

Interest Probability	*1*	*2*	QA16:Occurrence Frequency of Code Smells (number of events of occurrence), QA16:Change proneness of the modules in which the code smells reside
Priority	*2*	*7*	QA14:Class Score, QA14:Change frequency score, QA14:Severity score i.e. negative impact (measured by size, cohesion, coupling, complexity etc.), QA14:Number of code smells present in the class Threshold value, QA17:Smell Intensity, QA17:Metric thresholds