This version of the article has been accepted for publication after peer review but is not the Version of Record and does not reflect post-acceptance improvements or any corrections. The Version of Record is available online at: http://dx.doi.org/10.1007/s00766-024-00424-3.

Modelling the Quantification of Requirements Technical Debt

Judith Perera[1*], Ewan Tempero[1], Yu-Cheng Tu[1], Kelly Blincoe[2]

[1]School of Computer Science, The University of Auckland, New Zealand.
[2]Department of Electrical, Computer and Software Engineering, The University of Auckland, New Zealand.

*Corresponding author(s). E-mail(s): jper120@aucklanduni.ac.nz;
Contributing authors: e.tempero@auckland.ac.nz;
yu-cheng.tu@auckland.ac.nz; k.blincoe@auckland.ac.nz;

Abstract

Requirements Technical Debt (RTD) applies the Technical Debt (TD) metaphor to capture the consequences of sub-optimal decisions made concerning Requirements. Understanding the quantification of RTD is key to its management. To facilitate this understanding, we developed a conceptual model, the *RTD Quantification Model (RTDQM)*. Our work is grounded in the literature found via a Systematic Mapping Study (SMS) and informed by prior work modeling the quantification of software code-related TD types.

The key finding is that although RTD is similar to code-related TD in many aspects, it also has its own components. RTD can be incurred regardless of the presence of code-related TD. Unlike code-related TD, RTD has a feedback loop involving the user. RTD can have a cascading impact on other development activities, such as design and implementation, apart from the extra costs and efforts incurred during Requirements Engineering (RE) activities; this is modeled by the RTD Interest constituents in our model.

The model was used to compare and analyze existing quantification approaches. It helped identify what RTD quantification concepts are discussed in the existing approaches and what concepts are supported by metrics for their quantification. The model serves as a reference for practitioners to select existing or to develop new quantification approaches to support informed decision-making for RTD management.

Keywords: Requirements Technical Debt Quantification, Systematic Mapping, Conceptual Model

1 Introduction

Requirements play a crucial role in developing a software product — gathering requirements right and delivering the product that offers the best value to its end-users is essential for the success of a software product [1], [2]. Nevertheless, practitioners performing Requirements Engineering (RE), system design, and implementation activities can make sub-optimal decisions concerning requirements while performing their tasks. Some instances of sub-optimal decisions related to requirements are the inadequate gathering of user needs, ambiguous specifications, a wrong prioritization of requirements, inadequate (i.e., partial or incorrect) implementation of essential user needs, and design decisions that do not essentially satisfy user requirements. These sub-optimal decisions may be deliberate or even inadvertent but introduce consequences to the project. The consequences of sub-optimal decisions made concerning requirements are captured by the metaphor **Requirements Technical Debt (RTD)** [3], a metaphor borrowed from the Technical Debt (TD) literature.

Unless reduced or managed, RTD can have a negative impact on the project, leading to large cost overruns and potentially a bad reputation for the company due to not meeting customer satisfaction. However, to reduce or manage RTD, there must be a way to quantify it [4]. We define an approach that allows quantification (i.e., a *quantification approach*) as: "An approach that discusses concepts and metrics that could support RTD management decision-making." Our goal is to improve existing quantification approaches and provide guidance to develop new quantification approaches. For this, we need to better understand RTD quantification and identify gaps in the existing quantification approaches.

We conducted a Systematic Mapping Study (SMS) and established a theoretical foundation to improve the understanding of RTD quantification by formally defining RTD and through the development of a conceptual model, the **Requirements Technical Debt Quantification Model (RTDQM)**. This paper presents the results of our evaluation of existing quantification approaches, firstly through the SMS and secondly through comparing and analyzing existing approaches based on the conceptual model that we developed during our SMS.

Our initial work was reported in our RE conference paper [3], where we presented the results of the initial mapping study on RTD quantification and developed an initial version of RTDQM. This paper extends our previous work in the following ways:

- The SMS is updated and improved, adding 11 new papers to the initial 7 previously reported, resulting in 18 papers in the final dataset. However, they reinforced the same finding reported in our previous work [3] — there is no commonly agreed definition nor a commonly agreed way to quantify RTD.
- The initial model is updated to accommodate the newly identified concepts (19 new model concepts). Table 9 shows which concepts were added newly. We re-traversed the 7 initial papers to find evidence for the new model concepts that emerged from the 11 new papers.
- We expand the initial model to include RTD quantification for the non-functional aspect of requirements. Table 9 also shows which concepts were added newly for modeling the nonfunctional aspect (12 out of 19 concepts were added to model

the non-functional aspect). Doing so shows the relationship between non-functional requirements TD and architectural design.
- We combined RTDQM with previous work by Perera et al. [5], who developed a model for quantifying software code-related TD types (discussed in Section 7.3) to have a more complete model that captures both forms of TD. The combined model shows the cascading impact of RTD on other development activities, such as implementation and design, where code-related types of TD are introduced (RTD can also impact testing. However, this is not considered in the scope of this paper).
- We demonstrate how RTDQM can be utilized (discussed in Sections 8.1 and 8.2, respectively). Firstly, we show how the conceptual model can be utilized to compare and analyze existing quantification approaches to identify gaps in existing approaches based on model concepts. Secondly, we show how practitioners could select existing approaches or develop new approaches that fit their quantification needs based on the model concepts. An existing approach can be selected, or a new one can be developed based on which concepts and metrics are relevant for practitioners in a specific project (and on whether existing approaches support quantifying these concepts via metrics).

The remainder of the paper is organized as follows. Section 2 provides background and discusses related work, Section 3 and Section 4 report on the Methodology and Results of the improved SMS, respectively. Section 5 provides a high-level discussion based on the SMS. Section 6 reports on the development of the conceptual model, adding more information to what was initially reported in our conference paper [3], and Section 7 describes the resulting conceptual model (with improvements for capturing both functional and non-functional aspects of RTD) and the combined model that represents both RTD and code-related TD quantification models in the same model. Section 8 theoretically evaluates the model by demonstrating two examples of its utility: one to compare and analyze existing quantification approaches based on model concepts and the other to demonstrate how practitioners could select an existing quantification approach for their particular quantification needs (or develop new approaches). Section 9 compares code-related TD and RTD model concepts based on the combined model reporting on the observations. Section 10 discusses implications to researchers and practitioners and future work. Section 11 discusses threats to the validity of conducting the SMS and the model development. The paper is concluded in Section 12.

2 Background and Related Work

The following sections describe the terminology and background required to understand this paper and related work.

2.1 Terminology

A *conceptual model* in the context of our paper consists of elements that are abstract concepts and relationships between those concepts. The purpose of a conceptual model

is to capture the abstract concepts related to a phenomenon and illustrate the relationships between those concepts so that the phenomena can be well understood [6]. Furthermore, a conceptual model can serve as a reference to make inferences about concepts that could be modeled by the abstract concepts [7]. We have demonstrated this in Section 8.1 by utilizing our conceptual model (the Requirements Technical Debt Quantification Model) to analyze existing quantification approaches in the literature (see Sections 8.1, 8.1.1 and 8.1.2).

The Requirements Technical Debt Quantification Model (RTDQM) we developed in this work is a *conceptual model* that captures the phenomenon of "Requirements Technical Debt quantification" (Technical Debt quantification is described below in Section 2.2 while Requirements Technical Debt quantification is described in Section 2.3). The model development and the resulting model are described in the Sections 6 and 7 of this paper.

An *abstract RTD Quantification concept* in the context of our paper succinctly captures the notion described by multiple various Requirements Technical Debt (RTD) quantification-related concepts extracted from the primary studies. This is described further in Section 6 — model development methodology. An example of an abstract RTD Quantification concept in our model is the 'Cost of rectifying RTD.' This abstract concept captures multiple various concepts such as, *Cost to formalize and implement the neglected needs, Cost to fix the requirement smells within a SRS, Cost of comparing the current implementation with the set of possible changes, Cost of RTD payment, Cost of reducing or eliminating or resolving the obstacle and Cost to adjust problems caused by debt*, that were extracted from the primary studies.

Figure 2 in Section 6 shows the set of abstract concepts that resulted from our work and a few examples of various RTD quantification concepts extracted from the primary studies that are modeled by the abstract concepts. More examples from primary studies are discussed in Section 7, where the model concepts (i.e., abstract concepts) are discussed.

Metrics refer to some form of measurement that supports the quantification of a concept. For example, a concept may be 'Cost of rectifying RTD' while the metric for measuring the cost could be person-hours.

Quantification is a Technical Debt Management activity [8]. As defined by Li et al., measurement or quantification is the activity of analyzing and quantifying the costs and efforts required to assist in decision-making in TD Management (TDM). This is discussed further in Section 2.2.

The goal of our mapping study is to identify gaps in the quantification approaches published in the research literature. We define an approach that allows quantification, i.e., *a quantification approach* as: "An approach that discusses concepts and metrics that could support RTD management decision-making." In our work, the concepts could be not only costs and efforts but also benefits, priority, probability, or process/time-related (see Section 7).

2.2 Technical Debt (TD), TD Management (TDM) and Quantification

Technical Debt (TD) captures the consequences of sub-optimal decisions made during the development of a software product [5, 9]. These sub-optimal decisions could be made throughout the software development lifecycle, from requirements engineering to product maintenance [10]. Li et al. [8], identified that TD Management (TDM) is important in the implementation and maintenance of software. According to the authors, the TDM process includes activities such as Identification, Measurement (Quantification), Prioritization, Repayment, and Monitoring. They define measurement or Quantification as the activity of analyzing and quantifying the costs and efforts required to assist in decision-making in TDM.

The work of Perera et al. [5] identified during their SMS that quantification approaches (i.e., An approach that discusses concepts and metrics that could support TD management decision-making) proposed in the literature lack consensus. Therefore, the authors developed a conceptual model to model the quantification of code-related TD types such as Code, Design, and Architecture TD, based on findings from their mapping study and the work of Avgeriou et al. [11]. They theoretically validated their model by applying it to quantification approaches that were not initially used in the development of the model by conducting a second iteration of their mapping study and applying the model to the approaches quantifying CTD. Furthermore, the authors introduced an *Approach Comparison Matrix* to be able to compare and evaluate different quantification approaches based on their conceptual model [5]. In their future work and implications to researchers, they proposed investigating the quantification of non-code-related TD types which we chose to investigate in our study focusing on RTD quantification.

2.3 RTD, RTD Management (RTDM) and Quantification

Compared to code-related TD types, RTD is a relatively new area with less literature on the topic [4, 12]. Furthermore, there is no consensus yet if RTD can be considered a type of TD [13]. However, by observing the literature available, we are confident that it is a type of TD and that it needs to be given attention similar to any other type of TD since the consequences of RTD can similarly impact software development like any other type of TD. Few studies highlight the importance of managing RTD, e.g., Ernst [2], Lenarduzzi et al. [13], and Frattini et al. [14]. Below, we discuss the most relevant related work in RTD research. These studies are also discussed in Section 4.

Ernst [2] presents the *earliest definition* on RTD (in 2012) — *"the distance between the optimal requirements specification and the actual system implementation, under domain assumptions and constraints"*. According to Ernst, RTD describes tradeoffs regarding what requirements the development team ought to prioritize. The authors state that product value is measured in satisfied requirements and that RTD is incurred when requirements are prioritized in a way they do not deliver the most value to the customer [2]. Their tool, RE-KOMBINE, helps compare the current implementation to new proposed implementations. In their work, *quantification of RTD* is measured as the distance between the different proposed solutions and the current implementation.

Lenarduzzi and Fucci [13] provide a more comprehensive definition of RTD by extending the definition of RTD by Ernst to include upstream activities involving the elicitation of requirements and their translation into specifications. They give three definitions of RTD described as three types of RTD: Type 0, 1, and 2, based on Incomplete Users' needs, Requirement Smells, and Mismatch implementation, respectively. Furthermore, the authors propose strategies for the Identification, Quantification, and Payback of RTD.

Frattini et al. [14] emphasize the inadequacy of exploring the debt metaphor in the context of RE. As a result, they developed an initial analytical theory as a guide towards further research in the area. The authors organized concepts related to practitioners' understanding and managing of RTDM into 23 falsifiable propositions and additionally provide explanations to these propositions [14]. Their work is derived from interview and survey data. Our work, in contrast, is grounded in the existing literature found via a SMS and complements their work through the development of our model, RTDQM.

2.4 Secondary Studies on RTD Quantification

Although multiple secondary studies have been conducted for other types of TD, we found only one secondary study directly discussing RTD, a recent systematic literature review by Melo et al. [12]. The authors discussed the causes of RTD, strategies to identify and manage it, and metrics to support the measurement.

However, the 66 primary studies that resulted in their query did not exclusively focus on RTD. Hence, the metrics they were proposing for measurement were too based on prior work on other TD types rather than prior work on RTD. For example, one of the studies listed as 'quantification approach' in their study, the work by Nogroho et al. [15], is not focused on RTD.

Compared to their study, our SMS focuses on studies discussing RTD (i.e., TD related to Requirements) exclusively, and we specifically focus on the *quantification of RTD* based on findings from prior work on RTD. Primary studies included in our work extend the study period covered in Melo et al.'s work [12] (ours includes studies from 2020-2023 additionally). Our goal is different from theirs; ours was to conceptually model the quantification of RTD based on what we learned about RTD quantification from existing literature while theirs was to investigate which evidence helps to strengthen the TD requirements management process.

2.5 Definitions of RTD from Secondary Studies

We examined RTD definitions given in previous secondary studies where RTD was mentioned. Li et al. [8] and Melo et al. [12] use the early definition given by Ernst [2]. Alves et al. [16] refer to a similar definition; they define RTD as trade-offs made by the development team with respect to what requirements to implement and how to implement them. Requirements that are partially implemented, requirements that are implemented but not for all cases, and requirements that are implemented but do not fully satisfy non-functional requirements, such as security or performance, are given as examples for RTD by the authors.

Behutiye et al. [17] describe RTD as trade-offs in the requirements specification that are consequences of intentional, strategic decisions for immediate gains or unintentional changes in the context that have an impact on future costs of the project.

Definitions of RTD derived from the primary studies in our SMS can be found in Table 3 (and discussed in Section 4.2).

3 Systematic Mapping Study (SMS) — Methodology (RQ1)

We followed recommendations given by Kitchenham et al. [18] and Petersen et al. [19] to conduct the SMS, which answers the following research question:

- ***RQ1: What means of quantifying RTD are discussed in the literature?***

3.1 Search Strategy

We improved the search strategy of our SMS to include more papers in the final dataset compared to what was reported in our conference paper [3]. We improved our search query to be less restrictive by removing the list of keywords related to quantification and using the AND operation to combine phrases (e.g., "technical debt" AND "software requirements") instead of using specific phrases (e.g., "Technical Debt in Software Requirements"). As these improvements led to more results, we adjusted our inclusion/exclusion criteria to have E7 as a new exclusion criterion to reduce noise. The search phase was concluded in November 2023 (The search phase for the conference paper was concluded in March 2023; this paper extends this work). However, we did not limit our search to a particular period so as not to miss relevant papers.

In our initial work reported in our conference paper, we queried multiple digital databases: IEEEXplore, ACM, and Science Direct (based on recommendations by Brereton et al. [20]) and SCOPUS (following the recommendations by Cavacini [21]). Since all the primary studies finally analyzed in our initial execution of the SMS were available in SCOPUS (the most comprehensive database according to Cavacini [21]), we decided to query only SCOPUS for the work reported in this paper. However, we complemented the results obtained by querying SCOPUS with Backward Reference snowballing.

Listing 1 lists the final query string. Table 1 lists the Inclusion and Exclusion criteria.

```
TITLE_ABS_KEY(("requirements debt"
OR ( "technical debt" AND "requirements" )
OR ( "technical debt" AND "software requirements" )
OR ( "technical debt" AND "requirements engineering" )
OR ( "technical debt" AND "user stories" ))
AND (LIMIT-TO (DOCTYPE , "cp")
OR LIMIT-TO (DOCTYPE , "ar"))
AND (LIMIT-TO (LANGUAGE , "English"))
```

Listing 1 "Final Search String for SCOPUS"

Inclusion	I1 Discusses the quantification of RTD or provides some direction (or guidance) in quantifying RTD I2 Discusses quantifiable characteristics for RTD I3 Discusses metrics and tools concerning RTDM
Exclusion	E1 Not in English E2 Full text inaccessible E3 Talks, doc symposiums, posters, tutorials, proceedings, reports E4 Secondary or tertiary studies E5 Not RTD E6 Not peer reviewed E7 Discusses RTD but does not necessarily provide any direction (or guidance) in quantifying RTD

Table 1 Inclusion and Exclusion Criteria

3.2 Article Screening and Selection

Article screening and selection were performed following the *adaptive depth reading approach* suggested in Petersen et al. [19] starting from the title and continuing through the abstract, conclusion, and, finally, reading the full text. Additionally, we applied the inclusion/exclusion criteria listed in Table 1. The first author screened the articles, and two other authors evaluated the results at each stage of screening, including the stages where articles were screened by title, abstract, and keywords and based on the inclusion and exclusion criteria. Disagreements were discussed and resolved during iterative meetings. For example, the new Exclusion criterion E7 was added as a result of these discussions. The final set of articles that resulted from querying the digital database were discussed and agreed upon prior to performing reference snowballing on them. All articles, including those resulting from the snowballing rounds, were too discussed and agreed upon prior to data extraction and analysis.

Articles that described an approach to quantifying RTD or at least provided some direction in quantifying RTD were included. Articles that described quantifiable characteristics such as principal, interest, interest probability, uncertainty, or units of measurement in terms of time, cost, or effort were included. We also included articles discussing metrics and tools concerning RTDM. We excluded articles that were not in English, articles where the full text was inaccessible, that did not discuss RTD, and where RTD was discussed but nothing about quantification could be extracted from the article. Only peer-reviewed conference and journal articles were included as they are considered high quality.

3.3 Reference Snowballing

Backward reference snowballing [22] was performed on the final set of articles to complement the search query and to avoid the possibility of missing relevant articles. Backward snowballing was additionally guided by the related work sections in the articles. Articles that resulted from the snowballing were similarly screened using the inclusion and exclusion criteria while following the *adaptive depth reading approach* suggested in Petersen et al. [19]. Snowballing was conducted until there were no more

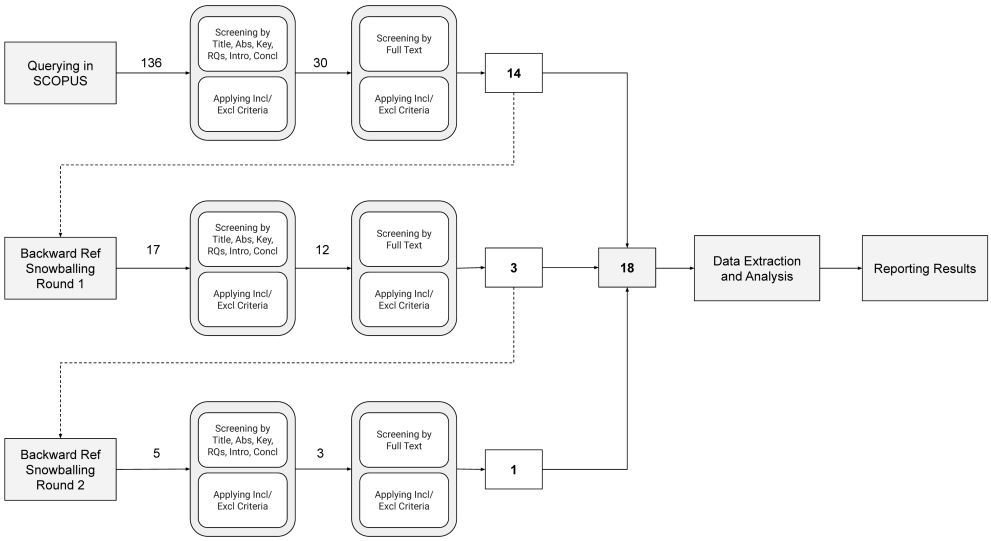

Fig. 1 Overview of SMS methodology and resulting articles at each stage

candidates for inclusion. The articles resulting from the snowballing were discussed and agreed upon by the authors prior to being included in the final dataset. The number of articles that resulted from querying and snowballing can be seen in Figure 1.

3.4 Data Extraction, Analysis, and Synthesis

The first author performed data extraction, analysis, and synthesis following recommendations from Petersen et al. [19] and Braun and Clarke [23] and discussed them with the other authors during meetings. Data extraction was done by following the *adaptive depth reading approach* to read the article's title, abstract, and conclusion, scanning for keywords, and then reading, in detail, the sections of the article where relevant information was found and finally, reading the full text. The *thematic analysis* approach recommended by Braun and Clarke [23] was followed for data analysis and synthesis. It is an effective method for identifying, analyzing, and reporting patterns and themes within data [23]. The high-level themes resulting from the analysis were Definitions of RTD, Concepts related to RTD quantification, Metrics, Tools and supported RTDM Activities, RTDM Strategies, Causes, Indicators, and Consequences of RTD, and Challenges associated with RTD. SMS results are reported under these themes in Section 4.

4 Systematic Mapping Study (SMS) — Results (RQ1)

Below, we report the results derived from 18 articles included out of the 158 articles obtained in total from combining the query results for SCOPUS and two rounds of Backward Reference Snowballing. Articles retrieved at each stage of the article

screening can be seen in Figure 1. The primary studies in the final dataset are listed in Table 2. Each primary study is referred to by a code P[n] throughout the paper. The codes for the papers can be found in the same table. Short descriptions of the papers can be found in Table A1 in the Appendix.

4.1 Demographics

Publications resulting from the SMS range from 1997 - 2023. We reported the study Title, Publication Year, Authors, Citation, Venue, and Citations received (in SCOPUS) in Table 2. The publication venues vary from different conferences to workshops. The oldest (1997) and the most cited (513 in SCOPUS) publication is by Karlsson et al. The next highest-cited article (147 in SCOPUS) is by Kazman et al., published in 2001.

4.2 RTD Definitions

The earliest definition of RTD was given by Ernst [2] (P7 in Table 2), which was discussed earlier in Section 2. Lenarduzzi et al. [13] (P2) introduced a more comprehensive definition, including 3 types of RTD incurred during upstream activities such as identification and formalization of requirements apart from the debt incurred during the implementation of requirements (Type 0, 1, and 2, based on Incomplete Users' needs, Requirement Smells, and Mismatch implementation, respectively). While they introduce two other types of RTD, 0 and 1, their type 2 is similar to Ernst's (P7) initial definition. Frattini et al. [14] (P10) uses the same definition as Lenarduzzi et al. [13] (P2). See Table 3) for RTD definitions from primary studies.

Abad et al. [4] (P1) define RTD as trade-offs in the System Requirements Specification (SRS) that result from intentional, strategic decisions or unintentionally due to changes in the context.

Charalampidou et al. [38] (P3) define RTD as insufficient or incomplete requirements and outdated requirements, which are similar to type 1 in P2 and type 2 in P2 (and P7), respectively. Bonfim et al. [1] (P4) define RTD as failures in SRS, which again aligns with type 2 in P2 (and P7). Belle et al. [25] define RTD as a poor or partial implementation of requirements into features; this is also similar to type 2 in P2 (and P7).

Ojameruaye et al. (P9) [27] define RTD as missing, under-specified, or outdated Quality Requirement (QR) documentation. This aligns with the definition in P3 (and type 1 in P2). Similarly, the definition given by Costa et al. [32], consequences of poor software development (i.e., QRs that are inadequately addressed and usability issues), aligns with P3. Mendes et al. [33] (P14) describe RTD as problems in documentation such as missing, inadequate, or incomplete artifacts, which again aligns with P3 (and type 1 in P2).

Sivattian et al. [34] (P15) describe RTD as the degree to which the company meets its business goals or satisfies its customers aligning with P7. Barbosa et al. [35] P(16) describe RTD as the distance between the optimal specification and the actual system implementation and Requirements Documentation Debt as missing, inadequate, or outdated documentation aligning with both P7 and P3, respectively. Karlsson et al.

[37] define it as producing a system viewed as sub-optimal by its developers, customers, and users; this definition also aligns with P7.

Kazman et al. [36] (P17), describes the phenomenon as how well the architecture has been designed with respect to its quality attributes. This can be inferred as aligned with the notion of optimal vs sub-optimal.

Therefore, we see that most of these studies use variations of the initial definition provided by Ernst [2] (P7), while some definitions refer to type 1 in P2 [13] (and P3). Type 0 in P2 seems to be used only in their study currently. However, P2 provides the most comprehensive definition so far, and it is important to capture type 0 as well since RE activities begin by capturing User Needs (as further discussed in our model). However, we see that the aspect of intentional and unintentional decisions that is captured in Abad et al. [4] is yet missing in P2's definition.

4.3 Concepts related to RTD Quantification

We identified 286 concepts (in total, without duplicates) related to RTD quantification from the primary studies.

We categorized them into categories: *process/ time, cost, benefit, probability, and priority*, based on the high-level themes that emerged during Thematic Analysis [23] (further discussed in Section 6) and additionally informed by Perera et al.'s work [5].

We identified 155 concepts for the *Process/ Time category*, 72 concepts in the *Cost* category, 23 concepts in the *Benefit* category, four concepts in the *Probability* category, and seven concepts in the *Priority* category. However, 25 of the concepts were categorized as metrics.

The complete list of concepts and their categories can be found in our Replication Package[1], we do not report them here due to space limits. Figure 2 in Section 6 shows some examples of these concepts extracted from the primary studies.

Section 7 and Table 9 describe the abstracted set of concepts that went into the conceptual model developed for RQ2 as the result of the systematic coding performed adopting the Thematic Analysis approach [23] (abstract concepts are illustrated in Figure 2, the development of the model is discussed in Section 6). The abstract concepts and their high-level themes *process or time, cost, benefit, probability, and priority* formed the classification scheme that we used to analyze the existing approaches found in our mapping study later in Section 8.1.

[1]Replication Package: https://doi.org/10.5281/zenodo.10900222

	Title	Pub Year	Authors, Citation	Venue	Cit.
P1	Using real options to manage Technical Debt in Requirements Engineering	2015	Abad et. al, [4]	RE	18
P2	Towards a Holistic Definition of Requirements Debt	2019	Lenarduzzi et. al, [13]	ESEM	7
P3	Integrating Traceability Within the IDE to Prevent Requirements Documentation Debt	2018	Charalampidou et. al, [24]	SEAA	2
P4	Requirements debt: causes, consequences, and mitigating practices	2022	Bonfim et. al, [1]	SEKE	0
P5	A novel approach to measure confidence and uncertainty in assurance cases	2019	Belle et. al, [25]	REW	2
P6	On the perceived harmfulness of requirement smells: An empirical study	2020	Lenarduzzi et. al, [26]	CEUR	0
P7	On the role of requirements in understanding and managing technical debt	2012	Ernst et. al, [2]	MTD	27
P8	Systematic elaboration of compliance requirements using compliance debt and portfolio theory	2014	Ojameruaye B. et. al, [27]	REFSQ	11
P9	Towards optimal quality requirement documentation in agile software development: A multiple case study	2022	Behutiye W. et. al, [28]	JSS	11
P10	An initial theory to understand and manage requirements engineering debt in practice	2023	Frattini J. et. al, [29]	IST	1
P11	An industry experience report on managing product quality requirements in a large organization	2017	Mohagheghi P. et. al, [30]	IST	13
P12	Sustainability debt: A portfolio-based approach for evaluating sustainability requirements in architectures	2016	Ojameruaye B. et. al, [31]	ICSE	5
P13	Towards a Process to Manage Usability Technical Debts	2022	Costa A.F.F. et. al, [32]	ICPS	0
P14	Impacts of agile requirements documentation debt on software projects: A retrospective study	2016	Mendes T.S. et. al, [33]	SAC	20
P15	Linking the Selection of Requirements to Market Value: A Portfolio-Based Approach	2001	Sivzattian et. al, [34]	REFSQ	45
P16	Organizing the TD Management Landscape for Requirements and Requirements Documentation Debt	2022	Barbosa et. al, [35]	UMBC	8
P17	Quantifying the Costs and Benefits of Architectural Decisions	2001	Kazman et. al, [36]	ICSE	147
P18	A Cost-Value Approach for Prioritizing Requirements	1997	Karlsson et. al, [37]	IEEE Software	513

Table 2 Demographics: Publication Year, Authors, Citation, Venue, Num of Citations from SCOPUS, First part of the Table — 7 papers found in the initial execution of the SMS reported in our conference paper [5], Second part of the Table — 11 papers newly added to the dataset from improvements to the SMS by the work reported in this paper

	RTD Definition
P1	Discusses definitions borrowed from Alves et al. and Ernst et al., and give their own definition by explicitly incorporating the main characteristics of TD (time-dependent, interest, context/environment dependent) to those definitions: *"The trade-offs in requirements specification that are consequences of the intentional strategic decisions for immediate gains or unintentional changes in the context that have an impact on the future cost of the project".*
P2	Provides a *holistic definition* of RTD, defining three sub-types of RTD: Type 0, 1, 2, which include debt incurred during the identification, formalization, and implementation of requirements. *Type 0 — Incomplete or neglected Users' needs*: Incurred when User needs expressed in feedback channels are neglected; when not all user needs are captured within a channel and when one or more relevant channels are not considered. *Type 1 — Requirement Smells*: Incurred when a requirements engineer, business analyst, or developer formalizes user needs into the specification. e.g., Linguistic constructs that can indicate a violation of the ISO29148 standard for Requirements quality. *Type 2 — Mismatch implementation*: Captures the mismatch between stakeholders' goals framed in the SRS and the actual system implementation. Incurred when developers implement a solution to a requirement problem or when the requirements framed in the SRS changes while the implementation does not change accordingly.
P3	*Requirements documentation debt: Insufficient or incomplete requirements:* Pieces of specifications, e.g., Use cases, User stories, SRS), that are lopped either at low quality or do not describe the system under development, and *Outdated requirements:* Cases in which specifications have been developed at an appropriate level of quality in the early releases of the system but subsequently are not updated with new requirements or changes in existing ones.
P4	Failures in the SRS, characterizing the distance between the desired specification of requirements and the actual implementation of these requirements in the system
P5	Poor/partial implementation of requirements into a feature(s), and compromises made regarding the specific requirements by a development team with regards to what to implement and how to implement.
P6	Uses the definition of Type 1 in P2
P7	Tradeoffs on what requirements the development team ought to prioritize. The distance between the optimal solution to a requirements problem and the actual solution, with respect to some decision space.
P8	Result of neglected compliance when engineering requirements of software i.e., the difference between the cost of an ideal resolution tactic and the cost of the selected resolution tactic (aligned with P7).
P9	Missing, under specified or outdated QR documentation (Aligned with P3).
P10	Uses the definition in P2
P11	The gap between what can be achieved with the available resources and the hypothesized "ideal" environments where sustainability goals are successfully and completely realized (aligned with P7).
P12	Unfulfilled quality requirements, Under-performance, Poor documentation.
P13	Consequences of poor software development. e.g., quality requirements that are not adequately addressed, Usability issues (aligned with P3).
P14	Problems in documentation e.g., missing, inadequate, or incomplete artifacts.
P15	Tradeoffs have to be made in order to meet a degree of customer satisfaction relative to the employed resources. The degree to which the company meets its business goals or satisfies its customers is RTD (Aligned with P7).
P16	Requirements and Requirements Documentation Debt indicate shortcuts taken in software development projects, resulting in requirements partially implemented and with outdated documentation. *Requirements debt* - the distance between the optimal requirements specification and the actual system implementation e.g., requirements that are only partially implemented (aligned with P7) *Requirements Documentation debt* - associated with problems found in software project documentation (e.g. missing, inadequate, or outdated documentation). A type of documentation debt is requirements documentation debt, which affects requirements specifications, causing a mismatch between the stakeholder's needs and the software implementation (aligned with P3, P7).
P17	How well the architecture has been designed with respect to its quality attributes e.g., modifiability, performance, availability, usability i.e., non-functional requirements (aligned with P7).
P18	Producing a software system that developers, customers, and users view as sub-optimal (aligned with P7).

Table 3 RTD Definitions

	Supported RTDM Activities
P1	Quantification
P2	Identification, Quantification, Repayment
P3	Monitoring and Prevention
P4	Prevention, Reduction
P5	Quantification
P6	Reduction/ Prevention, Quantification
P7	Prioritization, Quantification
P8	Quantification, Repayment
P9	Prevention
P10	Detecting, Measuring (Quantification), Tracking, Remediation
P11	Identification, Quantification
P12	Reduction
P13	Identification, Prioritization, Estimation (Quantification), and Monitoring
P14	Quantification
P15	Quantification
P16	Prevention, Prioritization, Repayment
P17	Quantification
P18	Prioritization

Table 4 Supported RTDM Activities

4.4 Metrics, Tools and supported RTD Management Activities

We aimed to capture approaches that made an effort toward quantifying RTD. Five approaches (See Table 4) found in our SMS focused completely on this RTDM Activity (P1, P5, P14, P15, and P17). P2, P6, P7, P8, and P11 discussed RTD quantification along with other RTDM Activities such as Identification, Repayment, Prioritization and Debt Reduction or Prevention. P1, P2, P5, P7, P8, P11, P12, P13, P15 and P17 introduced metrics to quantify RTD (See Table 5).

	Metrics
P1	Standard deviation of the rate of return on the value of the selected requirements over time, Net value of the option, Conditional value of the option, Present value of each node, Risk-adjusted probability, Net present value of the existing options
P2	Principal, Interest
P5	INCIDENCE (or Weighted Assurance Confidence), Uncertainty (based on INCIDENCE) — to measure RTD incurred for a given feature
P6	Extra effort related to the harmfulness of Req. Smells
P7	Distance
P8	Criticality, Likelihood of the obstacle occurring
P11	Sharpe ratio of an "optimal" architecture, Sharpe ratio of an "selected" architecture, Criticality, Likelihood
P12	Priority of a req., SMART metric for quality of QRs, Covergae and fulfilment of requirements
P13	End user satisfaction, Usability heuristics
P15	Market Line (relationship between expected return and Beta - constant of proportionality)
P17	*Quality attribute response measures* e.g., a particular level of performance measured by mean time to failure, level of reliability measured by steady-state of availability, Importance of QA (i.e., benefit of QA), Cost of an Architectural Strategy (i.e., Expected cost of implementing each Architectural Strategy), Benefit of an Architectural Strategy (i.e., Expected benefit of each Architectural Strategy), Desirability metric for each AS — i.e., Cost benefit ratio of mean cost and mean benefit, QA Score given by stakeholder

Table 5 RTD Metrics

P3 and P7 introduced tools. P3 proposed a tool-based approach to enable debt Monitoring and Prevention while P7 introduced RE-KOMBINE, a tool that helps compare one implementation to new proposed implementations based on different prioritizations of requirements. P4 investigated the causes and effects of RTD and actions to minimize RTD within an agile context. Although some approaches did not directly facilitate RTD quantification, they provided some guidance to be able to quantify RTD. Hence, they were included in our study results.

However, these approaches focus on different aspects of quantification. For example, P1 applies real options theory to quantify the Net Present Value (NPV) of a SRS, and P5 measures the uncertainty of a given feature being supported by a system, while P2 proposes measuring Principal and Interest; Principal as the cost to formalize requirements, cost to fix Requirements Smells, and cost to compare the current implementation with a set of possible changes (also P7), and the Interest as the extra effort related to the current development stage and the harmfulness of Requirement Smells (also P6).

The conceptual model we developed answering RQ2 helps understand how these different concepts discussed in the different approaches map to the concepts of RTD quantification (See Sections 6, 7, and 8, all mappings are provided in the Replication Package[1].

4.5 Proposed RTD Management Strategies

Table 6 shows strategies for managing RTD proposed in the primary studies. P2 provides Strategies for identifying, quantifying, and repaying RTD Types 0, 1, 2. P3 describes how to avoid insufficiently specified requirements or outdated requirements and promotes verification of requirements by multiple stakeholders. P4 emphasizes the need to validate and manage requirements, while P7 emphasizes the importance of tracking requirements throughout the software lifecycle.

P9 [17] introduces the use of lightweight artifacts to document quality requirements such as the Given/ When/ Then template and having sprints dedicated for documentation to reduce the introduction of RTD. P12 [31] suggests using SMART requirements as a way to measure if the requirements are specified adequately, apart from involving all relevant stakeholders when specifying and prioritizing requirements. P13 [32] proposes an approach to systematically identify and manage UTD.

P11 [30] proposes identifying the optimal portfolio of architectural strategies to maximize value and minimize risks, while P18 [36] emphasizes that tradeoffs should be made prudently (i.e., acceptable trade-offs), with respect to quality, cost, and time-to-market.

P10 [14] provides prevention practices and repayment strategies for both Requirements Documentation Debt and Requirements Debt. Examples can be found in Table 6. P16 provides 26 prevention practices and 18 repayment practices; the most common practices are listed in Table 6.

[1]Replication Package: https://doi.org/10.5281/zenodo.10900222

	RTDM Strategies
P2	*Type 0 — Identification:* Leverage techniques for automatically classifying and summarizing user feedback and recommending new features based on it, *Quantification:* Consider that implementing a neglected need in a later stage can be more expensive and which components need to be changed to address the RTD, *Payback:* Once the neglected user need is identified, formalize and include it in the SRS *Type 1 — Identification:* (semi) automatic detection within SRS, *Quantification:* Consider different negative impacts that each requirement smell can have on activities relying on SRS, *Payback:* Removing a problematic language construct leading to ambiguity while maintaining the original goal of the SRS *Type 2 — Identification:* Identify based on approaches for traceability between SRS and source code, *Quantification:* Determine the amount of change between the current implementation and the SRS, *Payback:* Implementation of the best new solution matching the updated SRS
P3	Avoid insufficiently specified requirements through communication, avoid over-engineering requirements, limit the number of incomplete requirement specifications by assuring completeness of requirements through the verification of requirements by multiple stakeholders, alleviate accumulation of RTD due to lack of requirements to code traceability, prompt update of requirement specifications to prevent accumulation of outdated requirements
P4	*Practices to reduce or mitigate RTD:* Meeting requirements elicitation, helping requirements analysis, supporting the implementation of SRS, requirements validation, assisting requirements management
P7	Tracking requirements throughout the software development lifecycle
P9	Light-weight artifacts to document QRs such as the Given/When/Then template, Documenting sprints (allocating separate sprints for documenting QRs), QR check lists, QR training, Documenting QR quality targets and decisions
P10	*Prevention Practices for Requirements Doc Debt* — Commenting code, Creating tutorials on how to fill documentation, Defining process and good practices for documentation, Defining roles concerning the documentation process, Documenting the project since it begins. *Prevention Practices for Requirements Debt* — Well-defined requirements, Following project planning, Following well-defined project process, Well-defined scope statement, Good allocation of resources in the team. *Repayment Practices for Requirements Doc Debt* — Adopting TD payment prioritization criteria, Keeping documentation updated, Reviewing outdated documentation *Repayment Practices for Requirements Debt* — Code refactoring, Monitoring and controlling project activities, Design refactoring, Investing effort on TD repayment activities, Changing project scope.
P11	Avoiding inappropriate selection of architectures that are not value- and risk-driven and debt-aware, identifying the optimal portfolio of architectural strategies with a total value that minimises risks and maximises returns.
P12	Moving to SMART requirements, involving the business stakeholders in the work for specifying and prioritizing quality requirements, understanding costs drivers when making tradeoffs
P13	Systematic identification and management of UTD
P16	*Most common prevention practices (26 in total)* — well-defined requirements, following the project planning, and following well-defined project process *Most common repayment practices (18 in total)* — code refactoring, monitoring and controlling project activities, and design refactoring investing effort on TD repayment activities, and changing project scope
P17	Incorporating economic models of software, that take into account costs, benefits, and risks to analyze software architecture for decision-making to meet quality attribute response goals.
P18	Making acceptable tradeoffs among goals such as quality, cost, and time-to-market.

Table 6 RTDM Strategies

4.6 RTD Causes, RTD Indicators and Consequences of RTD

Similar to any TD type, RTD can be incurred or caused intentionally or unintentionally. See Table 7, for causes and indicators. Inefficiencies in identifying and estimating Requirements (P2, P11, P16), inefficiencies in the Requirements specification or Requirement Smells (P2, P3, P10), and sub-optimal implementation of Requirements (P2, P3, P10) can happen due to negligence or deliberate choice to not

	Causes and Indicators of RTD
P2	Neglecting user needs, Missing to capture User feedback from one or more user feedback channels, Ambiguities introduced during formalization of requirements (Requirement smells), Implementing a sub-optimal solution to a requirements problem
P3	Insufficient and incomplete requirements missed due to the lack of requirements-to-code traceability, Documentation inefficiencies occurred intentionally or unintentionally (e.g., selecting not to apply a rigorous documentation process, documents are not sufficiently maintained due to tight schedules, developers not documenting requirements properly due to time limitations), Inconsistent management of requirements from different stakeholders
P4	Absence of a good requirements process may cause the RE steps to fail
P5	Requirements that are only partly implemented, Requirements that are implemented but do not support all cases, Requirements that are implemented but in a manner that does not entirely fulfill all desired non-functional requirements (e.g., security, performance)
P7	Inadequate or poorly conducted requirements elicitation and analysis
P8	Non-compliance e.g. , to information security compliance requirements
P9	*Causes* — Time constraints lead to under specification of QRs, Limited QR awareness, Communication gaps among team members *Indicators* — Missing or under specified or outdated QR documentation
P10	*Requirements Documentation Debt causes* — Deadline, The company does not give value to documentation, Non-adoption of good practices, Inaccurate time estimations,Inappropriate planning, Requirements not documented *Requirements Debt causes* — Deadline, Not effective project management, Changes in requirements, Inappropriate planning, High turnover of the team, requirements incompleteness, lack of formalization of requirements, not implementing requirements
P11	Causes — Poor estimates or misperceptions of system requirements, the system environment and external conditions, incomplete knowledge of the system, its environment and capacities *Indicators* — The final design may not have completely met the actual requirements, architecture does not fully contribute to the requirements imposed by sustainability, overdesign of the architecture so the potentials of the architecture are not fully utilized and the operational cost tends to exceed that of the generated benefits
P12	Failures, unimplemented functionality that were in the scope, Unfulfilled quality requirements, Underperformance, Poor documentation
P13	*Indicators* — Quality requirements that are not adequately addressed e.g., Usability requirements, Lack of usability standards, Inconsistencies between navigation aspects of the software, Violating relevant user interaction principals
P14	*Causes* — Lack of Information, Volatility of Requirements, Lack of Non-functional Requirements Identification, Lack of detailed specification, Problems in requirements documentation such as missing, incomplete or inadequate artifacts
P15	Poor selection of requirements for implementation
P16	*Causes (55 in total) categorized into: External factors, Development issues, Lack of knowledge, Methodology, Organizational, People, Planning and Management* — Deadline, not effective project management, change of requirements,inaccurate or complex requirement, and requirements elicitation issues, customer does not know his need, team's lack of knowledge and experience to develop the project, lack of well-defined process, inappropriate or poorly planned or poor executed test, and lack of requirements analysis, lack of qualified professional, lack of commitment and lack of team communication
P17	Changes to the existing architectural design

Table 7 Causes and Indicators of RTD

do so. P15 similarly discusses the poor selection of requirements for implementation. If a system does not satisfy one or more non-functional requirements, this could also be a cause of RTD (P5, P14).

Communication gap among team members (i.e., the lack of communication) was listed as a cause of RTD in both P9 and P17. Deadlines were another common cause of RTD, as discussed in P10 and P16. P9 discusses the same as 'time constraints'.

	Consequences of RTD
P3	Extra burden on maintenance tasks, inefficiency in project progress tracking, communication with customers on bug-fixing progress being hindered and testers not being aware of the requirements that need to be tested
P4	misunderstood, omitted, ill-defined, and poorly specified requirements in the RE phase and functional and non-functional requirements not being met in the implementation phase due to bad specification (i.e., not everything that was requested is delivered to the customer)
P7	Building the wrong product that does not meet customer satisfaction
P8	Consequence of non-compliance or extra cost if resolution does not fully mitigate the risk of obstacle, impact on compliance
P9	Consequences of missing or outdated QR documentation such as lack of common understanding of QRs and incorrect implementations, rework and additional iterations, TD accumulation, Informal quality management process
P10	*Requirements Documentation Debt* — Low maintainability, Delivery delay, Rework, Low external quality, Inadequate, non-existing or outdated documentation *Requirements Debt* — Delivery delay, Rework, Financial loss, Low external quality, Low maintainability
P11	Impact on sustainability dimensions e.g., on technical sustainability (i.e., impact of alternative architectures on sustainability goals and the likely debt due to the partial or poor satisfaction of these goals)
P12	Architectural impact of the QRs, further growth of TD that is due to under-performance
P14	Impact of New requirement (lack of info or lack of non-functional documentation), Impact of changed requirement (volatility of requirement)
P16	*Consequences or effects (33 in total) categorized into: Development issues, External Quality, Organizational, People, Planning and Management* — Delivery delay, rework, increased effort, low external quality, and low maintainability, design changes, inadequate documentation, and constant need for retest, project not completed, need for refactoring, bad code, stress with stakeholders, team demotivation, and stakeholder dissatisfaction, financial loss, impaired company image
P17	Architectural decisions can affect more than one quality attribute resulting in multiple non-functional requirements not being met (i.e., Architectural decisions might have consequences for several QA concerns)
P18	Developing a sub-optimal product

Table 8 Consequences of RTD

The volatility of requirements (i.e., requirements changes) was discussed as another common cause in studies P10, P14, and P16. P17 discussed changes to the existing architectural design, which can also cause requirement changes.

Consequences of RTD extracted from the primary studies are listed in table 8. P3 states that consequences can take the form of an extra burden on maintenance tasks. Furthermore, incomplete or insufficient requirements could lead to inefficiency in project progress tracking, hindered communication with customers on bug-fixing progress, and testers not being aware of the requirements that need to be tested. P4 describes that the absence of a good requirements process may cause the RE steps to fail, and that will generate consequences such as misunderstood, omitted, ill-defined, and poorly specified requirements in the RE phase and functional and non-functional requirements not being met in the implementation phase due to bad specification (i.e., not everything that was requested is delivered to the customer). P7 states that poorly conducted requirements elicitation can lead to building the wrong product that does not meet customer satisfaction. P18 states the same.

P9 discusses rework and additional iterations as consequences of RTD. P10 and P16 also discuss delivery delay, rework, low external quality, and low maintainability for both Requirements Debt and Requirments Documentation Debt. P16 lists 33 consequences in total. Table 8 lists the most common among them.

P8 mentions the impact of non-compliance, while P11 discusses the impact on sustainability dimensions as a consequence of RTD due to the poor satisfaction of non-functional requirements. P12 similarly discusses the architectural impact of quality requirements.

4.7 Challenges associated with RTD

Primary studies discuss various challenges in the field of RE associated with RTD. Common themes are as follows.

There is yet no consensus in the research community on whether RTD is a type of TD (P1, P2, P6), lack of formalization of RTD (P1, P2, P6), the inherent uncertainty and complexity in Requirements (P1, P7, P9, P16, P17) — e.g., uncertainties about customer (or market) requirements, project context and environment, and the feasibility, cost, and duration of developing each requirement [4], and difficulties in monitoring and traceability of Requirements (P3, P7, P10) — e.g., the effort of keeping requirements updated is high since it is difficult to identify which requirements can be affected when the source code is changed [38].

Prioritizing functional requirements over non-functional requirements can also be a challenge, according to P9. Software process models can be helpful, but they usually prioritize other activities than debt assessment and remediation (P10, P14). Furthermore, the user's satisfaction level can also vary, i.e., stakeholders may have different expectations (P8, P11), and stakeholders may prioritize requirements differently (P15, P16). Another challenge is that only a few teams usually track Requirements according to P7 [2].

For non-functional requirements, the design problem parameters are not completely known at the beginning of a project (P11, P12). P12 states that Specifying measurable requirements with verification scenarios is a challenge. Eliciting quality requirements and measuring and tracking them can be more difficult than their functional counterparts, according to P12. Furthermore, changes in non-functional requirements can also affect the system architecture (P15). The level of uncertainty and risk associated with design decisions and the difficulties in estimating costs and benefits are discussed as challenges associated with non-functional requirements in P17.

Our study contributes to some of these challenges by establishing a formal definition of RTD and a theoretical foundation for RTD quantification.

5 SMS — Discussion

This section provides a high-level discussion and a conclusion for the results obtained from the SMS. Section 8.1 provides a classification of the SMS dataset and findings based on the conceptual model for RTD quantification developed in the rest of the paper. Threats to the validity of the SMS are discussed later in Section 11.

5.1 A definition for RTD

The primary studies that resulted in our SMS discussed different instances where RTD could incur and in what form. For example, some studies discussed the incurrence

of RTD during the identification of requirements (e.g., incomplete user needs), while some discussed the incurrence of RTD during the documentation of requirements (e.g., incomplete or outdated or inadequate documentation, Requirement Smells), and the rest discussed the incurrence of RTD during the implementation of the system (e.g., poor or partial implementation of requirements, distance between the optimal SRS and the system implementation). Some studies also discussed that RTD could be incurred unintentionally or intentionally — due to strategic reasons. Some studies discussed missed, or outdated, or under-specified Quality Requirements (QRs) or how well a system architecture satisfies QRs.

Based on the various definitions found in the literature, drawing together the commonalities found in them, we synthesized the following definition for RTD:

> ***A Definition for Requirements Technical Debt (RTD):*** "RTD captures the consequences of sub-optimal decisions made concerning requirements, either deliberately (for strategic gains) or inadvertently (due to changes in context), during the identification, documentation, and implementation of requirements as features or architectural design decisions."

5.2 RTD Quantification (RQ1)

The goal of conducting the SMS was to identify what means of quantifying RTD are discussed in the literature. However, the primary studies that resulted in our SMS discuss different aspects of RTD quantification, and there is no common consensus on how to quantify RTD.

Furthermore, we did not find a common model that conceptualizes RTD quantification that can be used as a reference point to objectively compare and analyze the existing approaches in the literature to find out what the existing approaches support and do not support in terms of RTD quantification. This also creates difficulty for a software practitioner in choosing an appropriate approach for their RTD quantification needs.

This implies the need to unify the knowledge regarding RTD quantification and to be able to objectively compare and analyze existing approaches to find gaps in them. The rest of the paper addresses this problem via RQ2. We first develop a conceptual model, which we use to compare and analyze existing approaches (see Section 8.1) and provide guidance on developing new quantification approaches (see Section 8.2).

The conceptual model establishes a theoretical foundation for RTD quantification. The model development is discussed in Section 6, and the resulting model is discussed in Section 7. A theoretical evaluation is conducted in Section 8, effectively demonstrating the comparison and analysis of existing quantification approaches based on the model. This analysis also allowed the identification of gaps in the existing literature in terms of what model concepts are supported by the existing approaches.

Section 8 furthermore discusses how the model can be useful in selecting existing quantification approaches or developing new quantification approaches to support

informed decision-making for RTD management. Section 7 and 9 discuss further observations made from the model development, while Section 10 discusses implications for research and practice.

6 Modelling RTD Quantification — Methodology (RQ2)

The lack of unified knowledge on RTD quantification in the existing literature and the lack of a common conceptual model as a reference to compare and analyze existing approaches motivated us to develop a model to conceptualize the quantification of RTD. The development of our model answers the following Research Question:

- ***RQ2: How can we model the quantification of RTD?***
  - ***RQ2.1: What are the concepts sufficient to model the quantification of RTD?***
  - ***RQ2.2: What relationships can be identified among those concepts?***

The ***Requirements Technical Debt Quantification Model (RTDQM)*** (see Figures 4, 5, discussed in Sections 7.1, 7.2) was developed in part by examining the literature captured via our SMS and in part by examining what constitutes RTD quantification informed by its code-related counterpart in Perera et al.'s work [5], TDQM, a conceptual model developed to model the quantification of TD types related to software code such as Code, Design, and Architectural TD. Their work was also informed by literature captured via a SMS and by past models of TD, for example, Avgeriou et al.'s work [11].

We examined primary studies in our SMS to extract various concepts related to RTD quantification. As discussed in Section 4, 286 concepts (without duplicates) were extracted from the papers, which were then abstracted into 33 abstract concepts that went into the model for RTD quantification after multiple iterations of mapping related concepts into themes, high-level themes and examining them with reference to the TDQM concepts and their high-level themes. For this, we adopted the *Thematic Analysis* approach developed by Braun and Clark [23].

Thematic Analysis identifies patterns (themes) within the data and minimally organizes and describes the dataset in rich detail [23]. A theme captures something important about the data in relation to the research question, which is, in our case, abstract RTD quantification concepts (themes) and their high-level categories (high-level themes): *process or time, cost, benefit, probability, and priority.* The overview of the methodology is illustrated in Figure 2.

The systematic coding process we employed to map the concepts extracted from the literature (initial codes) to abstract RTDQ concepts (themes) and their high-level categories (high-level themes) is discussed in detail in Section 6.1. Section 6.2 provides an example.

First, we mapped the RTD quantification concepts extracted from the eleven new papers to the 14 model concepts that already existed from the initial work reported in our conference paper [3]. Any new concepts that could not be mapped to the existing model concepts went through the process of abstracting concepts into new

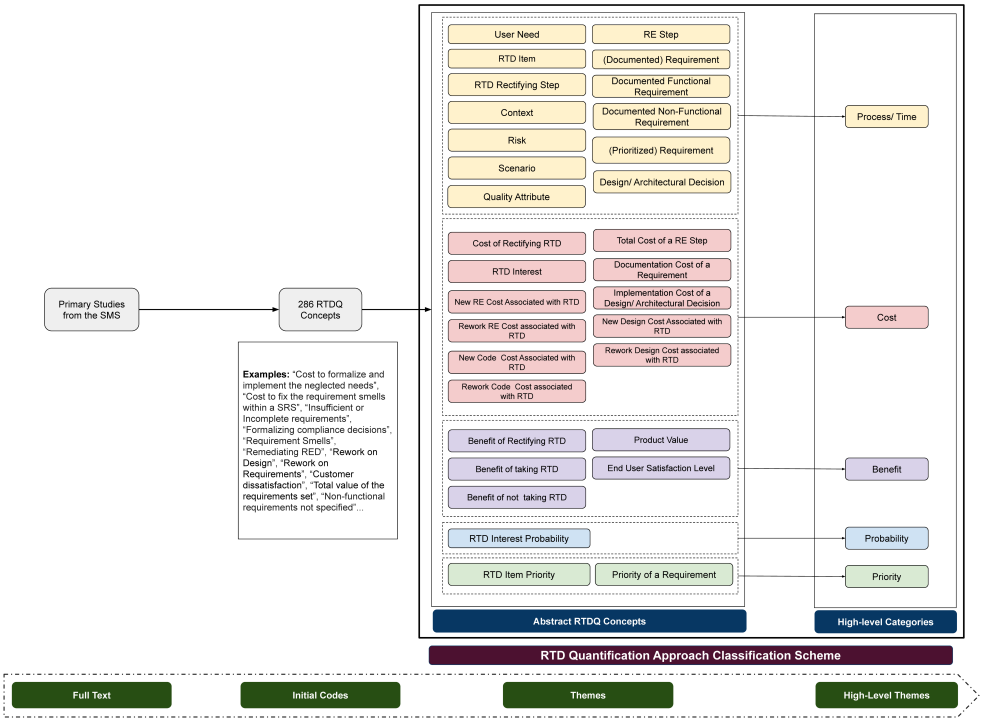

Fig. 2 Methodology for deriving RTDQM Concepts using *Thematic Analysis* adopted from Braun and Clark [23] — The RTD Quantification Approach Classification Scheme is used later in Section 8 to classify and analyze existing RTD quantification approaches

model concepts. The process was iterative until we found no new abstract concepts. Additionally, we re-iterated the seven initial papers to find evidence for new model concepts that emerged from the new papers.

The model reported in this paper is composed of 33 abstract RTD Quantification (RTDQ) concepts. Figure 2 shows the final set of 33 concepts. Table 9 also lists the final set of 33 RTDQM concepts and shows which concepts were added newly, from what was initially reported in [3]. The first part of the Table lists the model concepts reported in our initial work [3] while the second and third parts of the Table list what was added newly. Note that the concept 'RE Costs associated with RTD' was split into two concepts in this work, resulting in 'New RE Costs associated with RTD' and 'Rework RE Costs associated with RTD'. This is discussed in the results in Section 7.

Table 9 also provides two metrics we adopted from Junior et al. [39] to provide a hint on which concepts are grounded in the literature and how many sources support the concept. The density (d) metric shows how many sources support the concept — the number of papers that inform or validate a concept in our model. The groundedness (g) metric indicates how many excerpts or concepts (initial codes) extracted from a source (paper) are related to one RTD quantification concept in our model.

RTD Quantification Concept	Informed by Literature		Informed by TDQM
	density (d)	groundness (g)	TDQM counterpart
User Need	6	6	-
Requirements Engineering Step	9	9	Implementation Step
(Documented) Requirement	15	24	Feature
RTD Item	15	38	CTD Item
RTD Rectifying Step	7	9	CTD Rectifying Step
Total Cost of a RE Step	3	3	Total Cost of an Impl. Step
Cost of Rectifying (or remediating)	9	11	Cost of Rectifying
RTD Interest	14	45	CTD Interest
New Code Cost associated with RTD	3	6	New Code Cost associated with CTD
Rework Code Cost associated with RTD	5	8	Rework Cost associated with CTD
Benefit of Rectifying	1	5	Benefit of Rectifying CTD
Benefit of taking RTD	2	2	Benefit of taking CTD
RTD Interest Probability	5	5	CTD Interest Probability
Functional Requirement	7	10	-
(Prioritized) Requirement	0	0	Feature
New RE Cost associated with RTD	1	2	New Code Cost associated with CTD
Rework RE Cost associated with RTD	2	2	Rework Cost associated with CTD
Documentation Cost of a Requirement	0	0	Implementation Cost of a Feature
Benefit of not taking RTD	5	11	benefit of not taking TD
RTD Item Priority	5	5	CTD Item Priority
Priority of a Requirement	3	3	-
Non-Functional Requirement	10	14	-
Design Step	1	1	Implementation Step
Architectural/ Design decision	6	6	Feature
Context	9	13	-
Risk	5	6	-
Scenario	3	3	-
Quality Attribute	2	2	-
New Design Cost associated with RTD	0	0	New Code Cost associated with CTD
Rework Design Cost associated with RTD	2	2	Rework Cost associated with CTD
Implementation Cost of a Design Step	3	4	Implementation Cost of a Feature
End user satisfaction level	7	7	-
Product Value	5	5	-

Table 9 RTD Quantification Concepts (First column): First part of the Table — model concepts reported in our initial work [5], Second part of the Table — new concepts added to model the functional aspect of requirements, Third part of the Table — new concepts added to model the non-functional aspect of requirements, Density (d) - Num. of sources, i.e., SMS papers, Groundness (g) - Num. of concepts or excerpts extracted (initial codes) from all sources representing a RTDQM concept, Last column — TDQM concepts that informed the RTDQM concept

While some concepts for RTD quantification were informed by both the literature found in our SMS and TDQM, some concepts were informed by the literature only, and some were informed by TDQM [5] only (see Table 9). By utilizing TDQM as a reference, we could easily identify counterparts for RTD quantification.

6.1 The Coding Process

Adopting the terminology defined in Thematic Analysis [23], we define our *Dataset, Data Item,* and *Data Extract* as follows. Our *Dataset* is the set of 18 primary studies that resulted from the screening process of the SMS (See Figure 1). A *Data Item* is one primary study. A *Data Extract* is a chunk of data that has been identified within and extracted from a data item. This could be a phrase, a sentence, a paragraph, or a set of paragraphs in the full text of the primary study.

The first author identified and extracted various RTD quantification concepts as *Initial Codes* (see Figure 2 for examples) from the primary studies. The first author then mapped these initial codes extracted from the primary studies to a set of abstract RTDQ concepts (i.e., themes) that were identified as recurring themes among the various RTD quantification concepts extracted from the primary studies. *An abstract RTD Quantification concept succinctly captures the notion described by multiple various RTD quantification concepts extracted from the primary studies.*

The first author traversed all the primary studies iteratively, extracting new concepts, mapping them to abstract concepts, and then re-traversing the primary studies when a new abstract RTD quantification concept was identified. Every time an existing abstract RTD quantification concept was not able to capture the concept extracted from a primary study, a new abstract concept was introduced. Every time a new abstract concept was introduced, it was discussed with the other authors.

Mappings between the extracted concepts (initial codes) and abstract concepts (themes) were examined by at least two other authors than the first author during the process. When there was disagreement, it was discussed and resolved during meetings. An example of a disagreement that occurred between the authors was regarding the concept 'Requirements Engineering (RE) Step'. However, the authors finally came to the conclusion that the model must model the real world as closely as possible but should not be too complicated. Therefore, it was decided that detailed steps of RE do not need to be modeled. All abstract concepts were discussed among all authors for agreement before they were finalized as model concepts. The final set of 33 abstract RTDQ concepts, agreed among all authors, can be seen in Figure 2 also listed in Table 9.

We could further categorize these abstract RTDQ concepts (themes) into high-level categories (high-level themes): *process/time, cost, benefit, probability, and priority*, similar to Perera et al.'s work [5]. These high-level themes were recurrent among the concepts extracted in our study for RTD quantification as well.

Relationships between the RTD quantification concepts in RTDQM were informed by TDQM, work of Perera et al. [5], as well as derived from the literature. For example, the three RTD types described by Lenarduzzi et al. [13] informed the relationships in our model between the concepts RTD Item and RE Step, and RTD Item and Implementation Step describing the introduction of RTD Items during these steps

(RE Step includes capturing User needs and producing Requirement specifications). Relationships related to costs and benefits were mainly informed by TDQM [5], for example, the relationships between the costs and benefits of rectifying a RTD Item and the Interest incurred by a RTD Item. We could identify similar relationships between RTDQ concepts.

We defined two levels of coding for the Concept mappings, Direct (D) and Inferred (I), to show how closely the paper concepts mapped to the model concepts (i.e., the abstract RTDQ concepts) — *'the degree to which a paper concept maps to an abstract concept'* since not all concepts derived from the papers precisely mapped to the abstract concepts, some inference had to be made in some cases based on the full text of the papers. A Metric (M) mapping was defined to indicate the mapping between *metrics* described in the paper and the model concepts if we could determine that the paper provided some form of measurement to quantify the model concept. A Relationship mapping (R) was defined to capture if two model concepts had a relationship between them. In some cases, some model concepts were directly related, while in some cases, they were indirectly related.

- **'Concept' mappings**
  - **'Direct' mapping — indicated by *(D)* in Figure 3**: The paper concept (initial code) corresponds exactly to the model concept.
  - **'Inferred' mapping — indicated by *(I)***: The paper concept refers to the model concept in some way but does not correspond exactly. i.e., it is inferred that the paper concept corresponds to the model concept.
- **'Metric' mapping — indicated by *(M)***: The paper discusses some form of measurement that contributes to the quantification of the model concept.
- **'Relationship' mapping — indicated by *(R)***: The paper discusses or indicates the relationship between two model concepts (or between a model concept and a paper concept corresponding to a model concept).

An example of the systematic coding process is discussed below in Section 6.2, with reference to Figure 3, which illustrates the detailed mappings performed for Ernst, P7 (notation — P[n]) [2]. See our Replication Package[1] for the detailed mappings performed for the rest of the primary studies.

6.2 An Example of the Mappings

Figure 3 illustrates the mappings between the concepts related to RTD quantification extracted from the primary study (initial codes) and abstract RTDQ concepts (themes) of an example primary study, Ernst, P7 [2].

In Figure 3, the rows in the matrix pertain to the ten initial codes from P7 listed as *C1 to C10*. Each initial code is mapped individually to the abstract RTDQ Concepts listed in the ***columns*** of the matrix, denoted 1-33. The legend for the paper concepts (initial codes C1 - C10) can be found below the matrix, while the legend for the abstract RTDQ concepts can be found on the right side of the matrix.

[1]Replication Package: https://doi.org/10.5281/zenodo.10900222

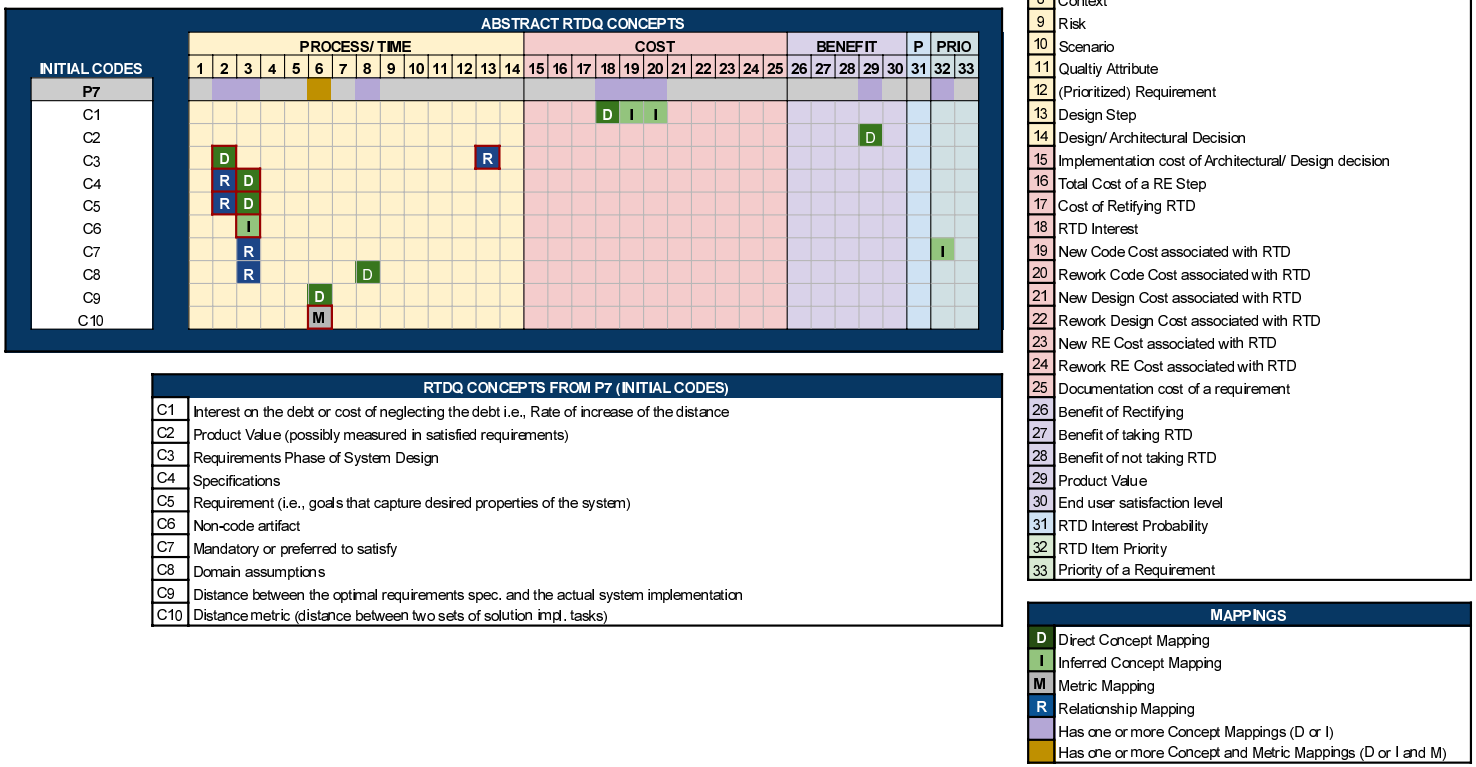

Fig. 3 Mappings performed for Ernst et. al., P7 [2]

Each cell in the matrix pertaining to the intersection of an initial code and an abstract concept is assigned a label for either a concept mapping 'D' or 'I' (according to whether the concept extracted from the primary study corresponds exactly to the abstract RTDQ 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 concept. A relationship mapping 'R' is used if there can be a relationship between the model concept the initial code is mapped to and another model concept listed in the columns of the matrix.

To describe this further (refer to the cells outlined in red in Figure 3), Ernst explicitly discuss the 'Requirements phase of system design (C3)'; therefore, the cell is labeled as 'D', indicating the direct mapping to the abstract RTDQ concept 'Requirements Engineering Step' (Column 2). In comparison, 'non-code artifact (C6)' is given mapping 'I' to the abstract RTDQ concept 'Requirement' (Column 3), indicating that we inferred this based on the text in the primary study. However, this is not the only mapping to 'Requirement' (Column 2) since the author explicitly discusses 'Requirement (C5)' and 'Specifications (C4)', which we have given mapping 'D'.

The 'Distance metric' (C10) is given mapping 'M' since it describes a form of measurement for the abstract RTDQ concept 'RTD Item'.

The relationship mapping 'R' between row C3 (which is mapped to column 2, 'RE Step') and column 13 ('Design Step') indicates that there can be a relationship between the concepts 'RE Step' and 'Design Step'. This relationship is illustrated in Figure 6 (an indirect relationship as variants of 'Development Steps' in the combined model of TDQM and RTDQM) later in Section 7. The relationship mapping 'R' between Row C4 'Specifications' (mapped as 'D' to 'Requirement') and Column 2 ('RE Step') indicates a relationship between 'Requirement' and 'RE Step'. Similarly, the relationship mapping 'R' between Row C5 'Requirement' (mapped as 'D' to 'Requirement') and Column 2 ('RE Step') also indicates a relationship between 'Requirement' and 'RE Step' (Column 2).

7 Modelling RTD Quantification: The Requirements Technical Debt Quantification Model (RTDQM) — Results (RQ2)

The Technical Debt Quantification Model (TDQM) developed by Perera et al. [5] captured the important concepts related to the quantification of code-related TD types and illustrated the relationships between those concepts. Modeling RTD quantification in our work resulted in the ***RTD Quantification Model (RTDQM)*** that captures important concepts related to RTD quantification and illustrates the relationships between them (See Figures 4, 5 and Table 9 — First column: RTD Quantification Concepts, Last column: code-related counterparts from TDQM [5]).

Evidence from the literature (mappings resulting from the coding process) for all model concepts and relationships can be found in our Replication Package[1]. The number of primary studies and the number of initial codes from the primary studies

[1]Replication Package: https://doi.org/10.5281/zenodo.10900222

supporting each model concept can be seen in Table 9. Below, we describe examples under each subsection describing parts of our conceptual model.

Note that RTDQM is presented in three steps as follows for ease of understanding of the reader:

1. Section 7.1 and Figure 4 presents **RTDQM** with the RTD quantification concepts and relationships modelling the *functional aspect of requirements*
2. Section 7.2 and Figure 5 presents the **complete RTDQM** with the RTD quantification concepts and relationships modelling the *functional and non-functional aspects of requirements*
3. Section 7.3 and Figure 6 presents **the combined model** where the complete RTDQM is combined with TDQM, Perera et al.'s work [5]. Through this, we can observe *how the impact of RTD flows through to the rest of the development stages.*

7.1 Modelling RTD Quantification

7.1.1 User Need, RE Step, Total Cost of a RE Step, (Documented) Requirement and how it connects to the Implementation of Features

See Figure 4. User needs (e.g., P2 User's needs [13], P3 stakeholder expectations [24]) for a software product that must be developed are usually captured through a Requirements Engineering (RE) Step (e.g., P7 Requirements Phase of System Design [2], P12 Specifying requirements [31]), which incurs a cost, the Total Cost of a RE Step similar to an Implementation Step incurring a Total Cost of Implementation in TDQM [5].

The RE Step produces one to many Documented Requirements (e.g., P1 SRS [4], P3 User stories [24], P6 Requirements specification [26], P6 A natural language formalization of a user need [26], P7 Specifications [2], P9 Documented Quality Requirement [28]). A Feature (the output of an Implementation Step in TDQM) is the implementation of either a single or multiple functional requirements according to Belle et al. [25]. This is indicated by the relationship between the 'Functional Requirement' and 'Feature' in Figure 4.

Non-functional requirements can also generate functional requirements or act as constraints on functional requirements (or feature implementation) [40] — we discuss this in Section 7.2 (illustrated in Figure 5).

7.1.2 RTD Items

Following the 16162 model [11], Perera et al. [5] modeled CTD as TD Items in TDQM. RTD could be modeled similarly as RTD Items. The difference is, a CTD Item is a software code artifact (e.g., Code Smell, Design Smell, Architectural Smell) whilst a RTD Item is a Requirement artifact (e.g., P2 Requirement Smell [13], P3 Outdated requirements [26], P2 and P16 Inadequately or insufficiently or incompletely implemented requirements [13, 35], P11 Inadequately satisfied non-functional Requirements [30], P9 Missing or outdated Quality Requirements [28]).

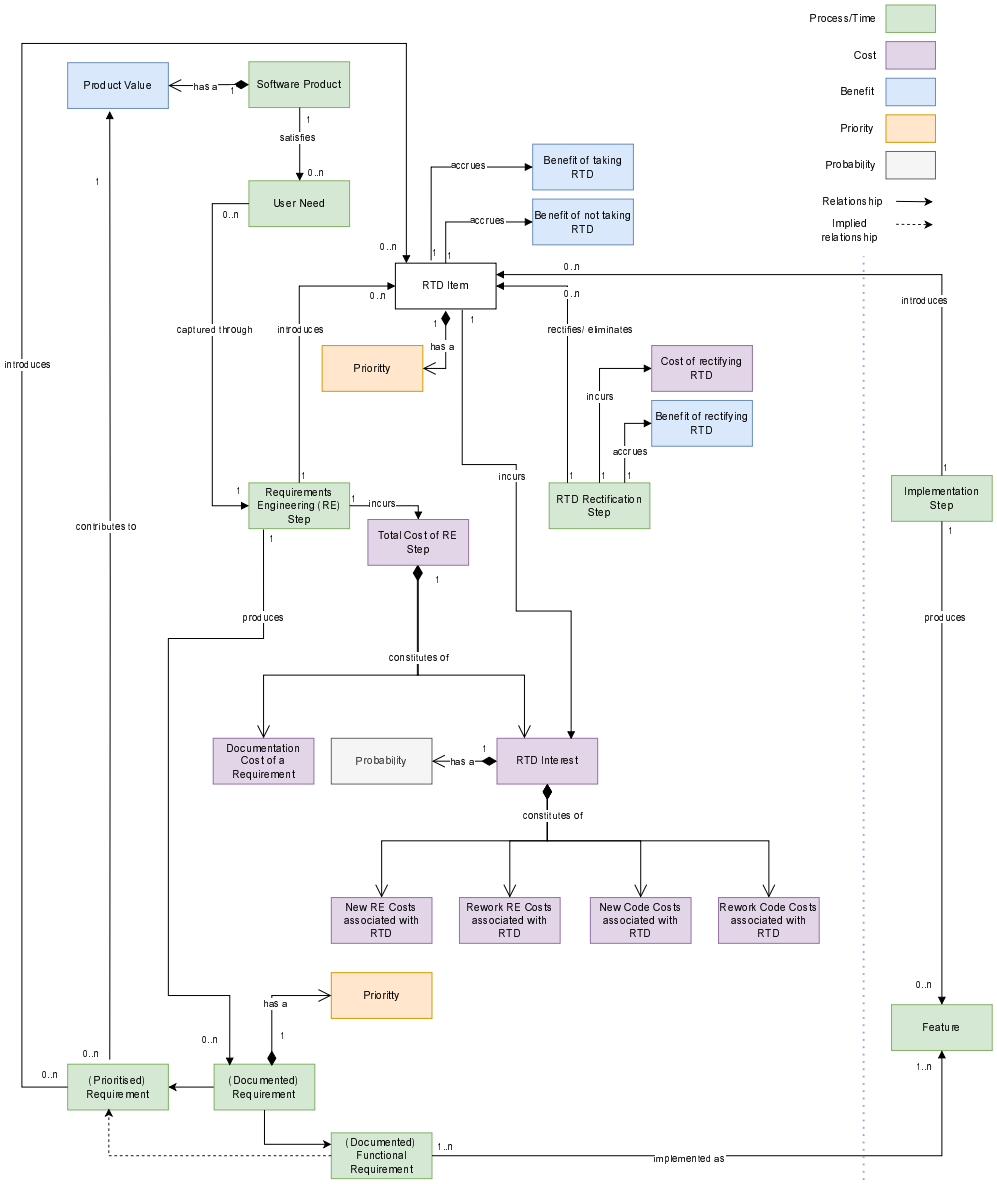

Fig. 4 RTDQM (functional aspect) — Concepts newly added can be found in Table 9, Right of dotted line shows the connection to TDQM, Perera et al.'s work [5] — the combined model is illustrated in Figure 6 and discussed in Section 7.3

Similar to CTD, RTD could also be introduced either deliberately or inadvertently, *either during RE* (e.g., when capturing User Needs or when Specifying Requirements in the Specification, illustrated by the relationship 'RE Step introduces RTD Item'

— These instances are described as RTD Types 0 and 1 by Lenarduzzi et al. [13] and by Charalampidou et al, as Requirements Documentation Debt [38]) *or during Implementation* (e.g., mismatch implementation, missing features, partially satisfied requirements, illustrated by the relationship 'Implementation Step introduces RTD Item'— described by Lenarduzzi et al. as Type 2, also by Ernst, Belle et al., and Abad et al. [2, 4, 13, 25, 26]).

7.1.3 RTD Rectifying Step and Cost of Rectifying RTD

In TDQM, a CTD Item can be eliminated via a Refactoring Step that incurs a Refactoring Cost. Similarly, a RTD Item can be eliminated or rectified via a RTD Rectification Step and incurs a Cost of Rectifying RTD. RTD can be rectified *either during RE* if identified early on before implementation *or during Implementation* if identified during that stage.

Lenarduzzi et al. [13] (P2) describe the Cost of Rectifying RTD as the cost to formalize and implement the neglected needs, the Cost to fix the Requirement Smells within a SRS, and the Cost of comparing the current implementation with the set of possible changes. Abad et al. [4] (P1) describe it as the Cost of exercising the option. Other examples include P4 Cost of RTD payment [1], P8 Cost of reducing or eliminating or resolving the obstacle (Principal) [41], and P14 Cost to adjust problems caused by debt [33].

7.1.4 RTD Interest and RTD Interest Probability

Similar to the extra or additional cost that is the consequence of the presence of CTD Items (i.e., TD Interest) a RTD Item can incur a RTD Interest that *could occur prior to the implementation phase*, e.g., as an extra cost to clarify an ambiguous formalization of a Requirement during Requirements Specification or as an extra cost to conduct additional interviews with Users (New RE Costs associated with RTD, Rework RE Costs associated with RTD) or *during the implementation phase*, e.g., as an extra cost to implement a workaround for a mismatch implementation (Rework Cost associated with RTD, New Code Cost associated with RTD) [4], [1], [2], [13]. Interest components are further described below.

However, there is a probability associated with the Interest since some requirements might not make a difference if unmet (e.g., P8 Likelihood of the obstacle occurring [41], P13 Estmiated Probability of Interest [32]). For example, RTD Items pertaining to an unused feature will not incur Interest [4]. This is modeled by the concept 'RTD Interest Probability'.

7.1.5 RTD Interest constituents

In TDQM, Perera et al. [5] decomposed the Implementation Cost of a Feature and CTD Interest into constituents, New Code Cost, and Rework Code Cost (i.e., costs incurred due to having to write new code or do rework). We saw the need to similarly decompose RTD Interest as well since RTD can also impact software code and there might be instances where new code must be written as well as instances where rework must be done. For example, if the Requirements change while the implementation is

done, then presumably, the code has to change, and some of those changes will require writing new code while some will require changing existing code. The existing code may have no CTD at all. Therefore, this cost differs from the CTD Interest constituents, New code cost associated with CTD, and Rework code cost associated with CTD. In other words, RTD Interest can occur regardless of the presence of CTD.

New Code Cost associated with RTD and the Rework Code Cost associated with RTD, model the cascading impact that can be incurred in the implementation phase due to RTD Items introduced in the RE phase or as the extra work to compensate for RTD introduced as a mismatch implementation during the implementation phase. Lenarduzzi et al. [13] (P2) discuss RTD Interest in terms of extra effort related to the current development stage, or the implementation of the selected change to address the amount of change between the current implementation and the SRS, which we see as examples for RTD Interest constituents New Code Cost associated with RTD and Rework Cost associated with RTD. Other examples for these RTD Interest constituents include P10 Rework on Code (extra rework due to RTD) [29] and P16 Need for refactoring [35].

RTD Interest can have other constituents, such as extra costs incurred during the RE Step that we initially referred to as 'RE Costs associated with RTD' in our previous work [3], e.g., the extra cost to perform additional interviews with Users in case the captured User Needs are incomplete. In this paper, we have represented the concept 'RE Costs associated with RTD' as two new concepts, 'New RE Costs associated with RTD' and 'Rework RE Costs associated with RTD', to illustrate the new and rework aspects of RE costs. Since Requirements are separate from code (i.e., RE is separate from Implementation), these costs are not represented by 'New code cost associated with RTD' or 'Rework code cost associated with RTD' concepts. Examples of these Interest constituents include P14 Impact of New requirement (impact of lack of information or lack of non-functional documentation) [33], P10 Rework on Requirements [29], P14 Impact of changed requirement (volatility of requirement) [33].

The extra costs in the RE phase could also occur due to a mismatch implementation (RTD introduced by the implementation) if this means having to do extra interviews with Users for feedback. Lenarduzzi et al. [26] (P2) refer to the cost related to the harmfulness of Requirement Smells. This can possibly incur all constituents of RTD Interest that we describe.

7.1.6 Benefit of Rectifying RTD, Benefit of taking RTD and Benefit of not taking RTD

Taking RTD can be beneficial in the short term (Benefit of taking RTD) as it allows faster delivery to market to gain a competitive advantage by delivering the most wanted set of Features, i.e., Minimum Viable Product (MVP). P10 describes this as the value for acquiring RTD, e.g., faster time to market [14]. Other examples include P1 Standard deviation of the rate of return on the value of the selected requirements over time [4].

However, accumulating RTD can be detrimental in the long run as it could impact the downstream activities, such as the software implementation phase. In such

instances, the developers might end up developing the wrong product for their customer that might have to be scrapped completely [2] (P7). Therefore, rectifying RTD early on would be more beneficial in the long run, during the early stages of the product development lifecycle prior to implementation. The benefit accrued by rectifying RTD is modeled as the 'Benefit of Rectifying RTD'. This was informed by the 'Benefit of Refactoring' in TDQM [5] (See Table 9).

Additionally, we modeled the concept 'Benefit of not taking RTD' since we encountered instances where this concept will be useful to quantify based on the evidence from the literature. P17 describes this as the benefit of an architectural design decision that does not introduce RTD [36]. Other examples include P1 Standard deviation of the rate of return on the value of the selected requirements over time [4], P11 Value of architectural design decisions [30], P15 Value importance, worth or desirability, [34] and P15 Expected Return (expected returns of the requirements) [34].

7.1.7 RTD Item Priority, Priority of a (Documented) Requirement and (Prioritized) Requirement

RTD Items can be prioritized for rectification or elimination (P7 Mandatory or preferred to satisfy [2], P8 Priority of an Obstacle [31], similar to CTD Items being prioritized for rectification or elimination in TDQM [5] — e.g., prioritized Code Smells are eliminated by refactoring software code.

Requirements can also be prioritized for implementation as Features (e.g., P12 Priority of a requirement [41], P15 Requirement's importance [34], P17 Importance of Quality Attribute [36]), resulting in the introduction of RTD if the wrong set of requirements is prioritized, i.e., sub-optimal prioritization of requirements [2, 27, 34].

7.2 Modelling the Non-Functional aspect of Requirements for RTD Quantification

Software requirements can be both functional and non-functional. Functional requirements are implemented as software Features while non-functional requirements are (usually) implemented as architectural or design decisions [32, 36]. Non-functional requirements can be constraints for functional requirements (i.e., constraints on a service or functions offered by the system) [40]. Non-functional requirements can also generate one or more functional requirements. This is indicated by the relationship 'may generate/ constraint' between Functional Requirement and Non-Functional Requirement in Figure 5.

For example, "Must remain confidential among authorized users." — This may appear non-functional. However, when the specification is developed in more detail, these requirements can generate functional requirements, such as the need to have a mechanism for user authentication [40]. According to Sommerville, non-functional requirements often apply to the software system as a whole and affect the system architecture rather than individual system components or features [40].

We modeled the concept 'Non-functional Requirement' (e.g., P4 Functional and non-functional requirements [1], P11 Sustainability Requirements [30]) and its relationship to 'Functional Requirement' (e.g., P4 Functional and non-functional

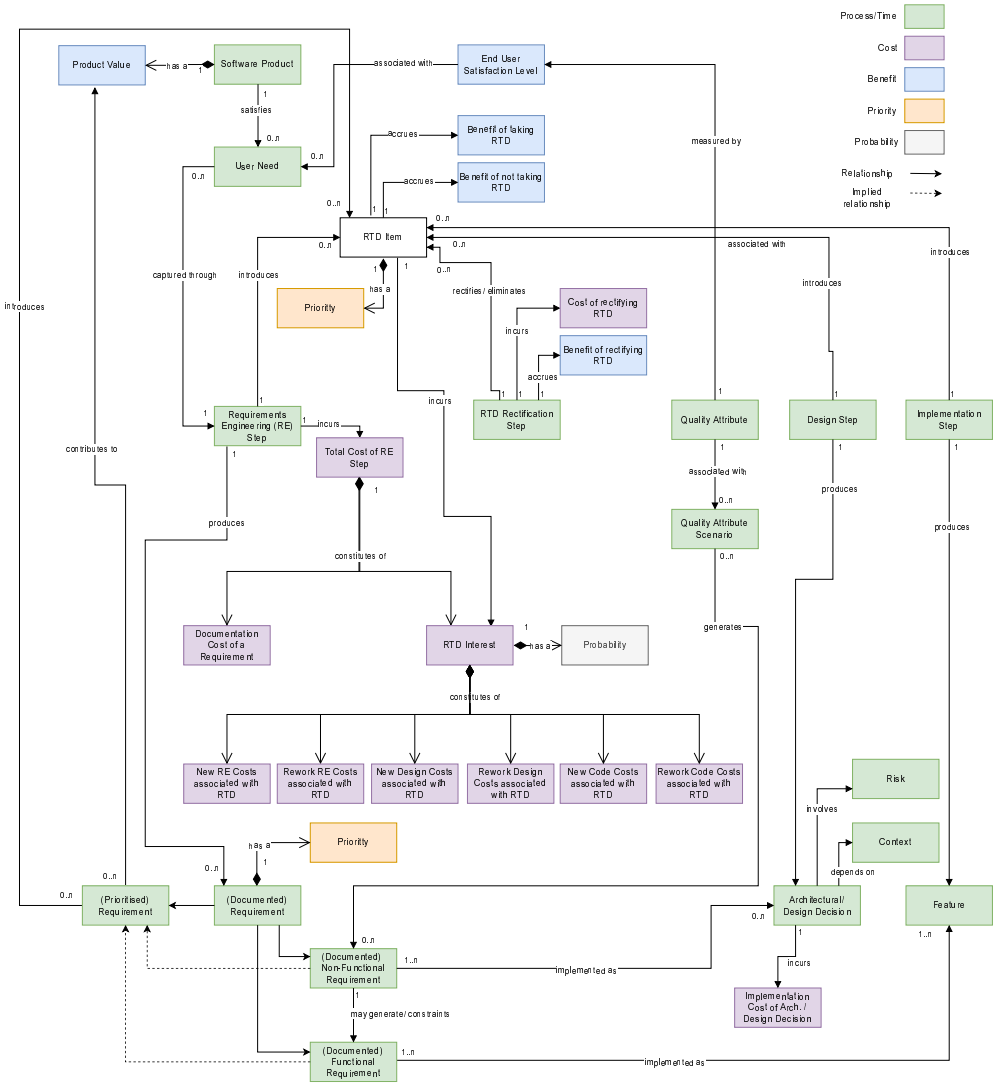

Fig. 5 Complete RTDQM (both functional and non-functional aspects) —Concepts newly added can be found in Table 9, RTDQM is combined with TDQM, Perera et. al's work [5] later in Figure 6 (and discussed in Section 7.3)

requirements [1], P11 Functional Requirements [30], P15 Functional requirement [34]) via 'may generate or constraint', based on Sommerville [40]. The concepts 'Design Step', 'Architectural/Design decisions', and 'Implementation Cost of a Design Step' were modeled to show the role of non-functional requirements in the architectural design phase by connecting the concept 'Non-functional Requirement' with the concept 'Architectural/ design decisions', based on Bass et al. [42] and the evidence from the literature (P17, P11, P8, P13, P8) [27, 30–32, 36].

A prioritized set of requirements contributes to the Product Value (P18, P15) [34, 37] and, in turn, to the level of end-user satisfaction when user needs are met. In the case of non-functional requirements, the level of end-user satisfaction (e.g., P8 Level of satisfaction [27], P13 End user satisfaction [32], P15 Customer satisfaction [34], P16 Stakeholder dissatisfaction [35]) can be related to, for example, the fulfillment of quality attributes such as performance, reliability, and availability (P17) [36, 42]. However, the ability of a software product to satisfy non-functional requirements is mostly context-dependent. It may involve the consideration of risk when making architectural or design decisions, for example, to satisfy quality attributes for a given quality attribute scenario (P11, P12, P17) such as meeting a performance threshold (P17, P11) [30, 36, 42].

The 'Design Step' can also introduce RTD when sub-optimal decisions are made regarding what non-functional requirements are prioritized to be satisfied by the architectural or design decisions and strategies (P8) [27]. For example, the level of end-user satisfaction regarding the system performance might not be met by a particular choice of design. This could lead to 'Rework Design Costs associated with RTD' (P10 Rework on Design [29]) and potentially 'New Design Costs associated with RTD' (P16 Design changes [35]) in the future if the design does not cater to the user's need (two of the Interest constituents of RTD). The relationship between RTD Item and Design Step, 'Design Step introduces RTD Item' was informed by Kazman et al.'s (P17) [36] and Costa et al.'s work (P13) [32].

The new concepts that were added to model the non-functional aspect of requirements can be seen in Table 9 (Third part of the Table). Figure 5 illustrates the complete model, incorporating the concepts for modeling the non-functional aspect.

7.3 A combined model of RTD and TD Quantification

The Technical Debt Quantification Model (TDQM) developed by Perera et al. [5] captured the important concepts related to TD Quantification for code-related TD and illustrated the relationships between those concepts. Modeling RTD quantification in our work resulted in the *RTD Quantification Model (RTDQM)* (discussed in Sections 7.1, 7.2) that captures the important concepts related to RTD quantification and illustrates the relationships between them. In this Section, we combine the two models to understand how RTD and code-related TD quantification fit together in the context of software development and what we can learn from that.

See Figure 6. A Software Product is usually delivered through multiple Releases. The releases are created through a Development Path that consists of a sequence of Development Steps. These Development Steps could be either Requirements Engineering, Design, Implementation, and Testing, for example. Note that our current model does not model all the development steps. For example, it does not model the Testing step.

TDQM was initially developed by Perera et al. [5] to model the types of TD related to software code, such as Design, Architecture, and Code TD. Therefore, our 'Design Step' in Figure 6 may have some overlap with the Design or Architectural TD discussed in Perera et al.'s work, which we plan to investigate further in our future

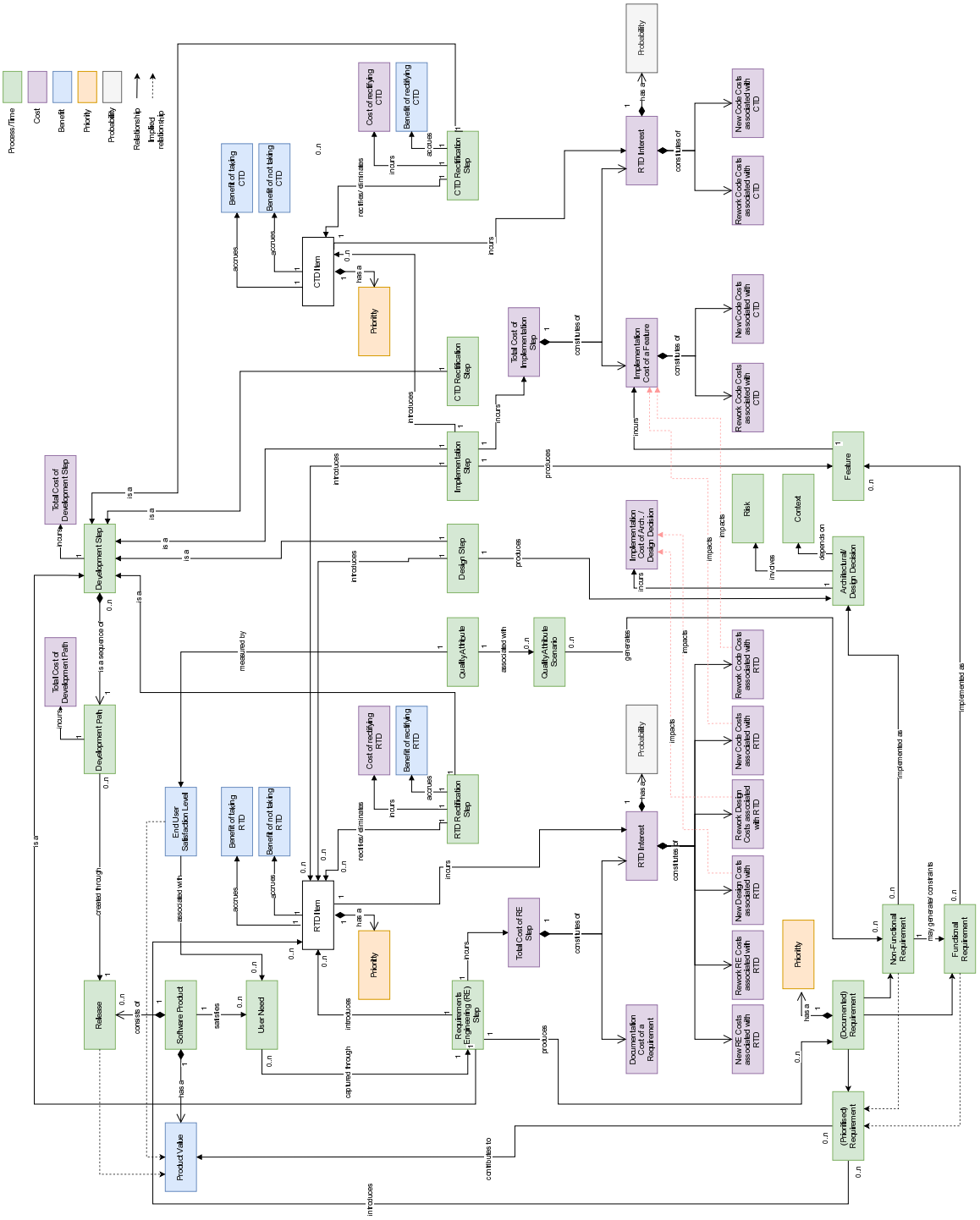

Fig. 6 The combined model of RTDQM (our work) and TDQM Perera et al.'s work [5]

work. In our current model, the 'Design Step' models the non-functional requirement aspect of RTD quantification.

Some companies could also have a dedicated Refactoring Step to refactor the software code or a Sprint to rectify the TD or RTD incurred. This is the reason for modeling RTD and CTD rectification steps parallel to the other development steps in Figure 6.

7.3.1 The cascading impact of RTD

By modeling the RTD quantification within the context of software development by combining the two models, we can observe how the impact of RTD flows through to the rest of the development stages.

The cascading impact of RTD on design and implementation costs is modeled by the 'impacts' relationships between the relevant RTD Interest constituents and the design and implementation cost concepts. For example, if problems with requirements specifications (i.e., ambiguities or Requirement Smells [13]) are not resolved early on, and they result in the misinterpretation of requirements, this can lead to the poor design of the system and, later on, require rework in the design. This is modeled by the 'impacts' relationship between the concepts 'Rework Design costs associated with RTD' and 'Implementation cost of an architectural/ design decision' in Figure 6.

In the case of functional requirements, the consequences of RTD could also impact the implementation, having to rework the inadequately implemented features, resulting in new code and rework costs associated with RTD. This is modeled by the 'impacts' relationship between new code and rework cost constituents of RTD Interest and the Implementation Cost of a Feature in Figure 6.

7.3.2 Feedback loop involving the User

Compared to artifacts related to software code, Requirement artifacts have a feedback loop involving the user to precisely capture User Needs in the RE Step (P2, P3, P6, P7, P16, P17, P18) [2, 13, 26, 36–38] and to meet the End user Level of Satisfaction with the Design and Implementation of the requirements (P8, P10, P11, P13, P15, P16, P18) [14, 27, 30, 32, 34, 35, 37]. Hence, the user is key in determining the optimal set of requirements of the software product that ultimately contributes to the Product Value. — 'contributes to' relationship between the (Prioritized) Requirement and Product Value. The satisfaction of a non-functional requirement, for example, a quality attribute, is 'measured by' its 'User Satisfaction Level', which is associated with the 'User Needs'. This illustrates the feedback loop involved with the user.

8 Evaluating the use of RTDQM

Our model, the ***Requirements Technical Debt Quantification Model (RTDQM)***, can be useful to both researchers and practitioners as a conceptual model that conceptualizes RTD quantification.

The model can serve as a reference point to compare and analyze existing quantification approaches and to develop new approaches in the instances where existing approaches may not satisfy practitioners' quantification needs. We theoretically evaluate these uses of the model in the subsections 8.1 and 8.2 below.

8.1 Comparing and Analyzing existing RTD Quantification Approaches

In this Section, we apply the RTD quantification approach Classification Scheme, which is based on the model concepts and their high-level themes: process/time, cost, benefit, probability, and priority, to compare and analyze existing approaches found in our mapping study. The classification scheme was illustrated in Figure 2.

Figures 7 and 8 illustrate the comparison of the 18 quantification approaches (a quantification approach, in the case of our paper, corresponds to one primary study) that resulted from our SMS in a matrix (adopted from Perera et al.'s *TDQM Approach Comparison Matrix* [5]) where the model concepts are in the numbered columns and the approaches are in the rows. The approaches P[n] are listed in Table 2.

Figure 7 ranks the quantification approaches based on how many model concepts a particular approach represents, while Figure 8 classifies the quantification approaches into those that discuss the functional and non-functional aspects of requirements.

The intersection of the columns and rows of the matrix in both Figures denotes whether a model concept is represented by the particular quantification approach, regardless of whether these concepts were initially mapped as 'D' or 'I' mappings during the development of the model (D and I mappings were discussed earlier in Section 6). Here, we pay attention to whether an existing RTD quantification approach discusses the model concept and whether it supports the quantification of that model concept by providing some form of measurement (i.e., metrics). Our model provides a common format to make comparisons by representing the existing quantification approaches via the model.

8.1.1 What do existing RTD Quantification Approaches quantify

Figure 7 ranks the existing quantification approaches based on how many model concepts are represented by a particular approach, which means that the particular quantification approach discusses these concepts. This ranking allows one to determine the most comprehensive (or the most complete) quantification approach in our dataset in terms of RTD quantification based on our conceptual model.

According to Figure 7, P11 represents 17 RTDQM concepts, which is the highest number of model concepts represented by a quantification approach in our dataset. Therefore, P11 is the approach that discusses RTD quantification most comprehensively from the quantification approaches in our dataset. P1 and P8 discuss 15 and 13 RTDQM concepts, respectively. Hence, P11, P1, and P8 are the top three most comprehensive RTD quantification approaches in our dataset.

Among the concepts discussed within the most comprehensive approaches, *Process/Time concepts:* (Documented) Requirement, Documented Functional Requirement, Documented Non-functional Requirement, and RTD Item, *Cost concepts:*

	Model Concepts
1	User Need
2	Requirements Engineering Step
3	(Documented) Requirement
4	Functional Requirement
5	Non-functional Requirement
6	RTD Item
7	RTD Rectifying Step
8	Context
9	Risk
10	Scenario
11	Quality Attribute
12	(Prioritized) Requirement
13	Design Step
14	Design/Architectural Decision
15	Implementation cost of Arch. / Design decision
16	Total Cost of a RE Step
17	Cost of Retifying RTD
18	RTD Interest
19	New Code Cost associated with RTD
20	Rework Code Cost associated with RTD
21	New Design Cost associated with RTD
22	Rework Design Cost associated with RTD
23	New RE Cost associated with RTD
24	Rework RE Cost associated with RTD
25	Documentation cost of a requirement
26	Benefit of Rectifying
27	Benefit of taking RTD
28	Benefit of not taking RTD
29	Product Value
30	End user satisfaction level
31	RTD Interest Probability
32	RTD Item Priority
33	Priority of a Requirement

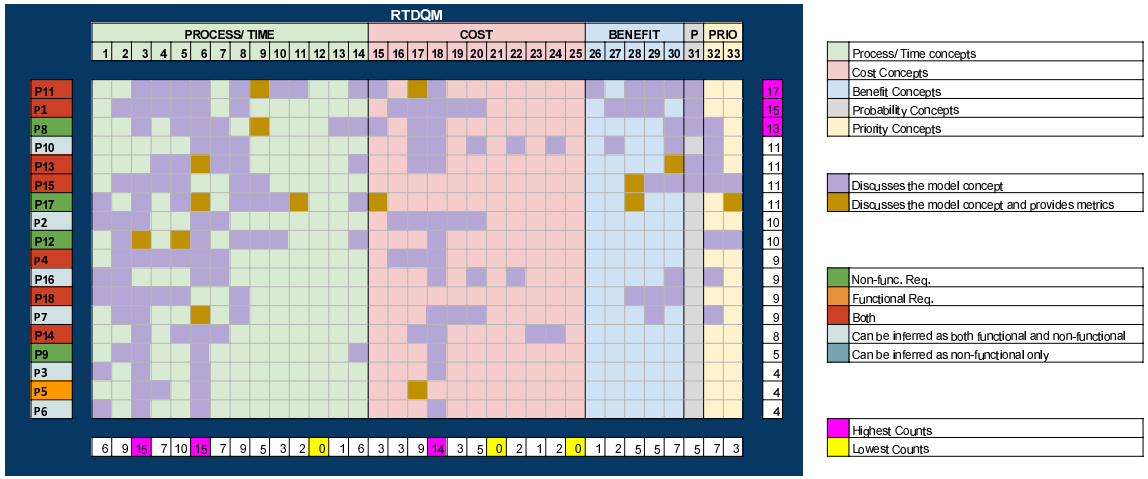

Fig. 7 Classification of existing approaches — Ranked based on the number of model concepts represented by a particular approach. Approach Comparison Matrix adopted from Perera et al. [5]. Papers referred by the codes P[n] can be found in Table 2.

Cost of Rectifying RTD and RTD Interest, and the *Priority concept:* RTD Interest Probability are discussed commonly between all three approaches P11, P1 and P8.

The row below the matrix provides a count of the number of quantification approaches that represent each individual RTDQM concept. This provides an overview of the concepts most commonly discussed among all quantification approaches, as well as which concepts are inadequately discussed among the existing approaches. In other words, this reveals gaps in the literature in terms of the RTDQM concepts discussed.

Model concepts most frequently discussed in the existing quantification approaches are 'RTD Item' (Column No. 6) and '(Documented) Requirement' (Column No. 3) since each of them has been represented by 15 approaches. 'RTD Interest' (represented by 14 approaches) has the next highest count. By observing the top 3 concepts, we can conclude that *most of the existing approaches found in our SMS were aware of the consequences of RTD (i.e., RTD Interest) and discussed its quantification.*

We observe that RTDQM concepts '(Prioritized) Requirement', 'New Design Costs associated with RTD', and 'Documentation Cost of a Requirement' are not represented by any of the quantification approaches. These concepts were informed by the TDQM model [5] during the development of our model RTDQM. Hence, they may not be represented in the quantification approaches in the literature (see Table 9 for what was informed by literature and what was informed by TDQM [5]). Documentation does require time and effort. Similarly, designing a new system in an instance where the existing system no longer caters to the user's needs has a cost. Our model shows that these concepts are worth investigating for RTD quantification.

In terms of Metrics that support the quantification of model concepts, we can observe that the following model concepts have one or more metrics or some form of measurement provided with, in the existing quantification approaches: 'Requirement', 'Non-Functional Requirement', 'RTD Item', 'Risk', and 'Quality Attribute' in the *Process/ Time* category, 'Implementation Cost of Architectural Design Decision', and 'Cost of Rectifying RTD', in the *Cost* category, and 'Benefit of not taking RTD', and 'End User Satisfaction Level' in the *Benefit* category, and 'Priority of a Requirement' in the *Priority* category.

However, Interest constituents (new and rework costs associated with RE, design and implementation phaces — Columns 19 to 24 in the matrix) identified through our conceptual model still lack metrics in the existing approaches. Surprisingly, 'Total Cost of a RE Step' and 'Documentation cost of a requirement' also lack metrics even though we expected that they may be quantified in the existing literature. *Benefit concepts:* 'Benefit of Rectifying', 'Benefit of taking RTD' and 'Product Value' do not have any metrics associated with them. Similarly, 'RTD Interest Probability' and 'RTD Item Priority' also do not have a form of measurement associated with them. This could be due to RTD yet being a relatively new area of research and the concept of RTD not being explored enough in the literature.

Summary of Findings:

- P11, P1, and P8 are the top three most comprehensive approaches.
- 'RTD Item', '(Documented) Requirement', and 'RTD Interest' are the most frequently discussed model concepts — This means that most of the existing approaches were aware of the consequences of RTD (i.e., RTD Interest) and discussed its quantification.
- 'Requirement', 'Non-Functional Requirement', 'RTD Item', 'Risk', 'Quality Attribute', 'Implementation Cost of Architectural Design Decision', 'Cost of Rectifying RTD', 'Benefit of not taking RTD', 'End User Satisfaction Level', and 'Priority of a Requirement' had metrics in the existing approaches.
- RTD Interest constituents, most of the benefit concepts, priority of a RTD item, and interest probability still lack metrics.

8.1.2 Functional vs Non-Functional Quantification Approaches

The matrix in Figure 8 illustrates the classification of the existing quantification approaches into those that discuss RTD related to the functional aspect of requirements and those that discuss RTD related to the non-functional aspect of requirements. Some approaches did not explicitly mention if they discussed functional or non-functional requirements. Still, we could infer from reading the complete article that some of them discussed both functional and non-functional aspects of requirements while some discussed only the non-functional aspect.

Based on the classification and with reference to the RTDQM concepts in the columns in the matrix, we can make observations about the RTDQM concepts represented by the different quantification approaches belonging to the categories: functional, non-functional, or both.

Considering the highest counts for the model concepts within the functional and non-functional classification, we can observe that for approaches discussing both functional and non-functional aspects, the model concepts '(Documented) Requirement', 'RTD Item', and 'RTD Interest' are the most commonly discussed (top three). They are the same as the top 3 concepts prior to classifying approaches into functional or non-functional (discussed in Section 8.1.1). However, this is different for approaches discussing non-functional only; the concept 'Design/ Architectural Decision' has been discussed more commonly than 'RTD Interest' while '(Documented) Requirement' and 'RTD Item' are similarly common among these approaches.

P5 is the only approach that discusses functional requirements only. The primarily discussed concepts for this approach in the *Process Time* category are; 'Requirement', 'Functional Requirement', and 'RTD Item'. For *Cost* concepts, 'Cost of Rectifying RTD' is quantified, but none of the other concepts are. P5 does not discuss 'RTD Interest', but this does not mean that the 'RTD Interest' is not quantified for functional RTD. All quantification approaches except P15, P17, and P18 discuss 'RTD Interest' regardless of whether they are discussing functional or non-functional requirements (P15 and P18 also discuss functional RTD).

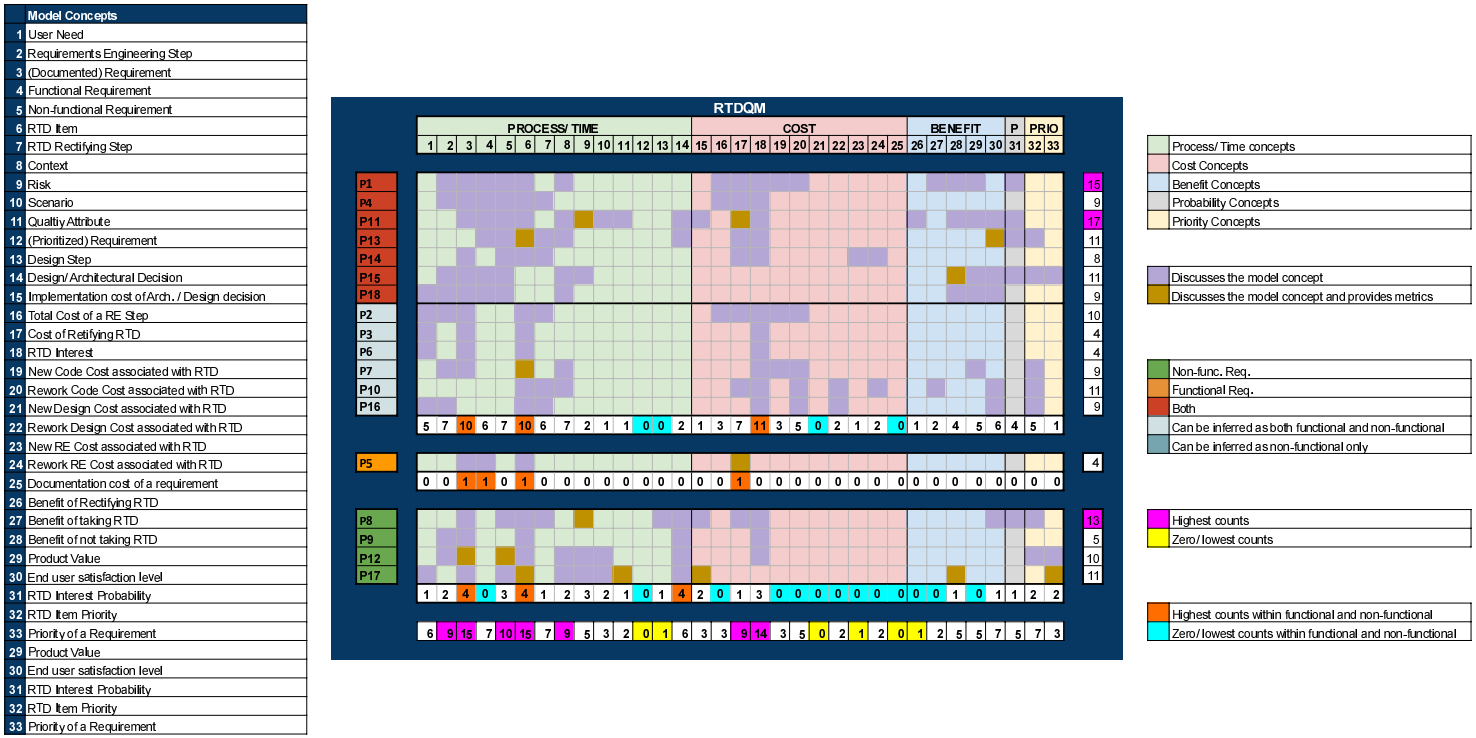

Fig. 8 Classification of existing approaches — Functional Vs non-functional. Approach Comparison Matrix adopted from Perera et al. [5]. Papers referred by the codes P[n] can be found in Table 2.

In terms of gaps in the literature, for the type of approaches discussing both functional and non-functional aspects, *Proces/ Time concepts:* '(Prioritized) Requirement', 'Design Step', *Cost concepts:* New 'Design Cost associated with RTD' and 'Documentation cost of a requirement' have not been discussed. Compared to the approaches discussing both the functional and non-functional aspects, 'Design Step' has been discussed among the approaches discussing the non-functional aspect only.

For approaches discussing the non-functional aspect only, *Proces/ Time concept:* 'Functional Requirement' is not discussed as expected. *Cost concepts:* 'Total Cost of a RE Step', 'Documentation cost of a requirement', and *Benefit concepts:* 'Benefit of Rectifying', 'Benefit of taking RTD', and 'Product Value' have also not been discussed in the approaches discussing the non-functional only.

Interest constituents: 'New Code Cost associated with RTD', 'Rework Code Cost associated with RTD', 'New RE Cost associated with RTD', 'Rework RE Cost associated with RTD', and 'Rework Design Cost associated with RTD' are discussed commonly in the approaches discussing functional and non-functional both but have not been discussed in the approaches discussing non-functional only.

Regarding metrics, model concepts 'RTD Item', 'Risk', 'Cost of Rectifying RTD', and 'Benefit of not taking RTD' have some metrics associated with them for the approaches discussing both functional and non-functional aspects as well as non-functional only. Model concepts '(Documented) Requirement', 'Non-functional Requirement', 'Quality Attribute', 'Implementation cost of Arch. / Design decision' and 'Priority of a Requirement' have metrics only in the non-functional only category. However, 'End user satisfaction level' does not have metrics for the non-functional only category, although the concept has been discussed in this type of approaches.

Summary of Findings:

- For approaches discussing non-functional only, the concept 'Design/ Architectural Decision' has been discussed more commonly than 'RTD Interest' while '(Documented) Requirement', 'RTD Item' are similarly common among all types of approaches.
- Regardless, they are categorized as approaches discussing functional or non-functional aspects; all approaches have no metrics associated with RTD Interest and its constituents.
- 'End user satisfaction level' does not have metrics for the non-functional only type of approaches.

8.2 Quantifying RTD to support informed decision-making

To demonstrate how the RTDQM model can help select existing quantification approaches or devise new quantification approaches for practitioners' quantification needs, we provide a simplified example based on a case study of an ongoing real-world project by which this work is funded[1].

[1]Veracity Lab: https://veracity.wgtn.ac.nz/

8.2.1 A software-intensive system supporting Organic Products Certification

In order to claim a product is organic, the process that produces it must be certified according to relevant standards. The example examines the requirements associated with a software-intensive system for supporting the certification process.

Full certification takes place over multiple years, and not all certification requirements need to be supported at the beginning of the process. This means a software company intending to develop a certification support system may choose to only support some of the requirements or only incompletely support some requirements in the early releases. This would provide the benefits of earlier time-to-market and allow the software company to gain experience with the domain.

That is, the company may choose to take on Requirements Technical Debt (RTD) with the expectation of accruing benefits by doing so. The company needs to make decisions regarding which requirements to omit or incompletely implement, and they would like to find a RTD quantification approach that would help support these decisions.

Ideally, the decision would be based on a RTD quantification approach that supports the specifics of the decision (specifics are discussed in Section 8.2.3 with reference to an example decision-making scenario related to the prioritization of requirements). We show how RTDQM can help choose such a quantification approach.

8.2.2 Requirements

Certification agencies, their auditors, the certification panel, and the organic producers are the end-users of this system. Once the organic producer applies for certification, an on-site audit follows the assessment of the application to audit the product's, process, and premises compliance with Organic Standards and Regulations. Once applied for certification, organic producers are considered clients of the certification agency.

During an onsite audit, an audit report will be produced with or without Corrective Action Requests for the Client in the cases of non-compliance. The client must perform the Corrective Actions before they can proceed with their operations in order to be eligible for the certification. Corrective action may take the form of, for example, changes to the ingredients of the product, or changes to the Organic Management Plan (OMP). If the certification agency approves that the Client adheres to the Organic Standards and Regulations, the certification is issued.

End-user needs can lead to both functional and non-functional requirements to be implemented in the software system. Below, we summarize a simplified subset of the requirements.

Functional Requirements

- **FR1:** The system must allow Clients to apply for certification by uploading all the data required to meet the requirements for an application. e.g., an Organic Management Plan (OMP) and other evidence, e.g., a Sector Map separating organic and other produce.

- **FR2:** The system state should change to "Awaiting on-site Audit Report" for a Client who is yet to pass an on-site audit.
- **FR3:** An auditor must be able to upload an Audit Report and issue Corrective Action Requests (CARs) for the Client.
- **FR4:** The system must have the functionality to import data from external applications e.g., soil testing applications, and have the functionality to view those reports.
- **FR5:** The Client should receive notifications on their mobile when a CAR is issued.

Non-Functional Requirements

- **FR6:** User authentication mechanism to ensure that the authorized person is viewing, entering, or modifying the data, i.e., a user is authenticated to carry out the operations.
- **FR7:** Encryption of data at the record level.
- **ADD1:** Layered Architecture implementing platform-level, application-level, and record-level protection of the data, e.g., system authentication at the platform level, database login authentication at the application level, and record access authorization at the record-level.

8.2.3 Prioritization of requirements

The Requirements prioritized to be implemented in the first **Release** can be either **Functional Requirements**, or **Non-Functional Requirements**, or both. Functional Requirements are implemented as **Features** while Non-Functional Requirements are implemented by **Architectural or Design Decisions**.

Consider that the following are two candidate sets of prioritized requirements. The company must select one of them to implement in the first release while knowing how much RTD they may incur and what costs and benefits may be associated with that.

- **Set1: FR1, FR3, FR6**
- **Set2: FR1, FR4, ADD1**

A 'sub-optimal prioritization of requirements' (i.e., RTD instance or the **RTD item** in this example) can lead to RTD interest payments. This could be in the form of RTD Interest associated with the Design and Implementation phases —- **New and Rework Design Costs** or **New and Rework Code Costs**. All these RTD quantification concepts are illustrated as **RTD Interest** constituents in our model, RTDQM.

Furthermore, there can be a cost to rectify the incurred RTD, that is, the **Cost of rectifying RTD** and a benefit associated with that, the **Benefit of Rectifying RTD**. There could also be a benefit accrued by taking on the RTD, that is, the **Benefit of taking RTD**. There could also be a benefit of not taking on RTD in the first place, that is, the **Benefit of not taking RTD**. All or a few of these concepts can be the *specifics* the company wants to consider when making their decision to take on or not take on RTD with their prioritization of requirements in the first **Release**.

Figure 7 shows what RTD quantification concepts are discussed in the existing quantification approaches and what concepts are supported by metrics in these

approaches. Table 10 provides the details of the metrics. Based on what RTDQM concepts (or combination of concepts) the company would want to make their informed decision for RTD management, they could select an existing quantification approach (or multiple complementing approaches) that would fit their quantification need.

If existing approaches do not support their quantification need, they could alternatively develop a new RTD quantification approach to fit their need. Our model helps identify such instances.

Identification of a suitable Quantification Approach

Consider an example where the company wants to make its decision based on how much **Cost of rectifying RTD** will be incurred in Release 2 for the RTD introduced by the selected prioritization of requirements in the first Release. Figure 7 indicates whether the existing approaches represent the concept **Cost of rectifying RTD** and whether they provide metrics for quantifying it. Specifics of the metrics can be seen in Table 10.

According to Figure 7, approaches P1, P2, P4, P5, P8, P10, P11, P13, and P14 discuss the concept **Cost of rectifying RTD**. However, only P11 and P5 provide some form of measurement. According to Table 10, P11 supports measuring the "sharp ratio of an optimal architecture" and the "sharp ratio of the selected architecture", which contribute to measuring the cost of moving towards the optimal architecture (essentially rectifying the RTD introduced in the first Release). P5 provides "confidence" and "uncertainty", which contribute to measuring the RTD incurred for a feature, i.e., how much it costs to fix the RTD. Therefore, the company can decide if they want to choose one of these approaches as fit for their purpose.

Identifying the need to develop a new Quantification Approach

As another example, consider that the company wants to make its decision based on how much **Rework code costs associated with RTD** may be incurred in the next release (or the next few releases) if the RTD incurred by the selected prioritization of requirements in the first release is not rectified for some time.

According to Figure 7, even though approaches P1, P2, P7, P10, and P16 discuss this concept, they do not necessarily provide a way to measure it (this is also indicated in Table 10). Therefore, this is an instance where new metrics or new quantification approaches may need to be developed.

9 Modelling RTD Quantification — Discussion

9.1 A Comparison of RTD and CTD

The development of RTDQM and combining it with Perera et al.'s work, TDQM [5] (See Figure 6 and Section 7.3), enabled a comparison between the quantification of code-related TD and RTD.

A comparison between the concepts for code-related TD and RTD quantification can be observed in Table 11 where we drew parallels between the RTDQM and TDQM concepts where possible. We discuss the main observations below. From here on, we

RTDQM Concept	Existing approaches discussing the concept	Existing approaches providing metrics for the concept
New Design Costs associated with RTD	-	-
Rework Design Costs associated with RTD	P10, P16	-
New Code Costs associated with RTD	P1, P2, P7	-
Rework Code Costs associated with RTD	P1, P2, P7, P10, P16	-
Implementation Cost of a Design/ Architectural Decision	P8, P11, P17	P17 - Cost of Architectural Strategy (i.e., Expected cost of implementing each Architectural Strategy)
Implementation Cost of Feature	P1, P15, P18	-
Total cost of a Development Path	P1, P2, P3, P5, P7, P11	P1 - Standard deviation of the rate of return on the value of the selected requirements over time, Net value of the option, Conditional value of the option, Present value of each node, Net Present value of existing options, Risk-adjusted probability
Benefit of taking RTD	P1, P10	-
Cost of rectifying RTD	P1, P2, P4, P5, P8, P10, P11, P13, P14	P11 - Sharpe ratio of an "optimal" architecture, Sharpe ratio of a "selected" architecture, P5 - confidence, uncertainty

Table 10 RTDQM Concepts and metrics to make an informed decision for prioritizing requirements — model concepts discussed in the example are in *italics*

will refer to code-related TD as CTD and that model as CTDQM to distinguish between RTD and CTD easily.

9.1.1 Similar Concepts in RTDQM and TDQM

RTDQM and CTDQM Concepts in the first part of Table 11 are similar concepts. We discuss below the concepts that were most commonly discussed in the RTD quantification approaches found in our SMS (see Figure 7 for the comparison of the occurrences of the concepts; highest counts are highlighted).

The most commonly discussed concepts were RTD Item and RTD Interest. A TD Item (either CTD or RTD) refers to a unit of Technical Debt, i.e., an artifact in the code or requirements that is a symptom of the incurrence of TD. TD Interest refers to the consequences of accumulating RTD or CTD.

CTD vs RTD Items

CTD Items include software code-related artifacts, while RTD Items can take various forms such as ambiguous, poorly or inadequately specified requirements that are

documentation-related artifacts, design trade-offs, and implementation that do not completely satisfy user needs.

Examples of CTD Items include software code-related artifacts such as Code Smells, Design Smells, and Architectural Smells [5].

RTD Items can take the form of ambiguous or low-quality requirements (P2, P6, P7), poorly specified requirements (P9), poor requirements documentation (P12), insufficient or incomplete requirements (P3), missing or inadequate or outdated requirements documentation or missing, incomplete or inadequate artifacts (P3, P9, P14, P16), or Requirement Smells (P2, P6), neglected or missed user needs (P2), inconsistencies in the SRS (P4), unspecified non-functional requirements (P14), inconsistencies in functional and non-functional requirements (P4), absence of requirements (P4), accumulated requirements in the backlog (P4), partially implemented requirements (P1, P5), and unmet requirements (P4), unfulfilled quality requirements (P12) and design trade-offs (P17) [1, 2, 4, 13, 24–26].

Like CTD Items, RTD Items could also be introduced into a software system inadvertently or through deliberate choice (P1, P3) either during RE activities (e.g., when capturing user needs, when specifying requirements in the SRS) or during design (e.g., design trade-offs, unfulfilled quality requirements) [31, 36] or during Implementation (e.g., mismatch implementation, missing features, partially satisfied requirements) [2, 4, 13, 24–26].

According to Abad et al. (P1) [4], trade-offs in requirements specification can be consequences of intentional, strategic decisions made in pursuit of immediate gains. Charalampidou et al. [38] state that documentation inefficiencies can occur intentionally by selecting not to apply a rigorous documentation process or unintentionally due to insufficient maintenance of documents, e.g., developers not documenting requirements properly due to time limitations.

CTD Interest vs RTD Interest

CTD Interest impacts the implementation and maintenance activities and does not impact upstream activities such as RE. New Code Costs associated with CTD and Rework Code Costs associated with CTD model the consequences of RTD related to the software code. In contrast, RTD Interest can impact both upstream (e.g., RE activities) and downstream activities (e.g., design, implementation or maintenance activities) of the software product development life cycle (P1, P2, P3, P4, P6, P7, P9, P10, P11, P12, P13, P14, P16) [2, 4, 13, 26, 38]. Therefore, RTD can have more Interest constituents relevant to other development stages, such as design and implementation, compared to CTD.

RTD can have additional consequences compared to the consequences of CTD, e.g., having to do additional interviews with Users to validate newly emerged requirements and having to clarify an ambiguous requirement in the specification; these involve costs associated with RE activities only (P2) [13], rework on requirements (P10), and the impact of changed requirements causing additional work in the RE phase (P14). New RE Costs associated with RTD and Rework RE costs associated with RTD concepts capture those costs.

Poorly conducted requirements elicitation (P4, P7) can also lead to building the wrong product that does not meet customer satisfaction [1, 2]. Bonfim et al. (P4) [1] describe this situation as unmet functional and non-functional requirements due to bad specification, and not everything requested is being delivered. New Design Costs associated with RTD and Rework Design Costs associated with RTD (P10, P16) concepts capture the costs that are consequences of sub-optimal design decisions related to the requirements.

RTD Interest incurred during the development stage (P1, P2, P7, P10, P16), e.g., to implement a workaround to compensate for a neglected user need, extra effort related to the current development stage (P7), cost of neglecting the debt (P7), rework on code (P10), and the need to refactor (P16), can be more expensive than resolving such issues earlier in the software development life cycle, which is the downside to accumulating RTD [2, 4, 13]. New Code Costs associated with RTD and Rework Code costs associated with RTD capture such consequences.

RTD can also lead to other issues such as inefficiency in project progress tracking (P3), hindered communication with customers on bug-fixing progress (P3), and testers being not aware of the requirements that need to be tested (P3), causing an extra burden for software maintenance tasks [38], lack of common understanding of QRs (P9), the architectural impact of QRs (P12), incorrect implementation leading to unhappy customers (P9), Informal quality management process (P9), low external quality, and low maintainability (P16), the constant need for re-testing (P16) and also delivery delays and thereby lead to stakeholder dissatisfaction and impaired company image.

9.1.2 Concepts specific to RTDQM

RTDQM concepts, User Need, Context, Risk, Scenario, Quality Attribute, Product Value, End User Satisfaction Level, Priority of a Requirement, and Non-Functional Requirement are specific to the RTDQM model (See Table 11).

The User is key in determining the optimal set of requirements for developing a software product to reach the desired product value that meets end-user satisfaction. Therefore, compared to artifacts related to software code, Requirement artifacts have a feedback loop involving the User to precisely capture User Needs, document them, and implement them as Features or Design decisions as discussed in Section 7.3.2. This is supported by evidence from the RTD quantification approaches (P2, P3, P6, P7) in the literature [2, 13, 26, 38]. This is not the case with the quality of the software code since it is an internal aspect that is not directly visible to the end user.

Furthermore, Requirements involve Risk and may be determined by Quality attributes and Scenarios, for example, Non-Functional Requirements, according to the evidence from the literature (P11, P17) [31, 36]. Therefore, such specific concepts are required to be considered in the context of RTD quantification.

9.1.3 Concepts specific to CTDQM

Concepts specific to the TDQM model from the work of Perera et al. [5] were: Release, development Path, Development Step, Total Cost of a Development Path, and Total

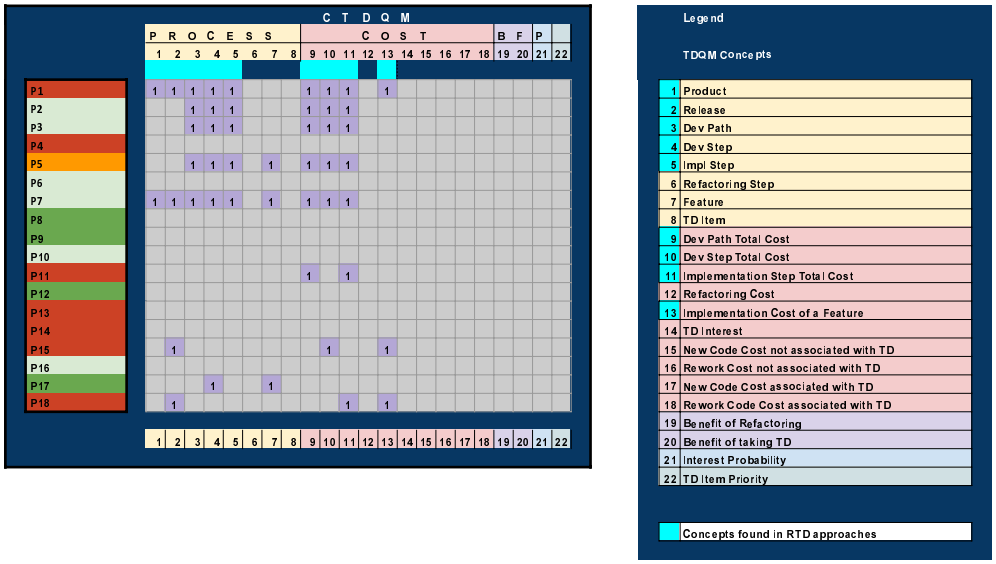

Fig. 9 CTDQM Concepts discussed in RTD Approaches — The Approach Comparison Matrix and the TDQM Concepts are from Perera et al.'s work [5]. Papers referred by the codes P[n] are listed in Table 2

Cost of a Development Step (See Table 11). We utilized the TDQM approach comparison matrix [5] to understand if these concepts were discussed in the RTD approaches as well since the concepts were related to the context of software development, although they were not initially modeled in our RTDQM model but in the TDQM model.

Figure 9 shows that the concepts Release, Development Path, Development Step, Total Cost of a Development Path, and Total Cost of a Development Step were represented by the RTD approaches as well. Hence, they are not specific to CTD quantification but are common to RTD quantification as well.

9.1.4 Concepts common to both models

Software Product was the only concept that was common to both the models (See Table 11). Figure 9 verifies that the concept is a common concept to both phenomenons, RTD quantification and CTD quantification.

RTDQM Concept	**Category**	**Parallel CTDQM Concept**	**Category**
Requirements Engineering Step	Process/ Time	Implementation Step	Process/ Time
(Documented) Requirement	Process/ Time	Feature/ Arch. or Design Decision	Process/ Time
Functional Requirement	Process/ Time	Feature	Process/ Time
RTD Item	*Process/ Time*	*CTD Item*	*Process/ Time*
RTD Rectifying Step	Process/ Time	CTD Rectifying Step	Process/ Time
(Prioritized) Requirement	Process/ Time	Feature/ Arch. or Design Decision	Process/ Time
Design Step	Process/ Time	Implementation Step	Process/ Time
Design/ Architectural Decision	Process/ Time	Feature	Process/ Time
Implementation Cost of Archi./ Design Decision	Cost	Implementation Cost of a Feature	Process/ Time
Total Cost of a RE Step	Cost	Total Cost of an Implementation Step	Cost
Cost of Rectifying RTD	Cost	Cost of Rectifying CTD	Cost
RTD Interest	*Cost*	*CTD Interest*	*Cost*
New Code Cost associated with RTD	Cost	New Code Cost associated with CTD	Cost
Rework Code Cost associated with RTD	Cost	Rework Code Cost associated with CTD	Cost
New Design Cost associated with RTD	Cost	New Code Cost associated with CTD	Cost
Rework Design Cost associated with RTD	Cost	Rework Code Cost associated with CTD	Cost
New RE Cost associated with RTD	Cost	New Code Cost associated with CTD	Cost
Rework RE Cost associated with RTD	Cost	Rework Code Cost associated with CTD	Cost
Documentation Cost of a Requirement	Cost	Implementation Cost of a Feature	Cost
Benefit of Rectifying RTD	Benefit	Benefit of Rectifying CTD	Benefit
Benefit of taking RTD	Benefit	Benefit of taking RTD	Benefit
Benefit of not taking RTD	Benefit	Benefit of not taking RTD	Benefit
RTD Interest Probability	Probability	CTD Interest Probability	Probability
RTD Item Priority	Priority	CTD Item Priority	Priority
User Need	Process/ Time	-	-
Context	Process/ Time	-	-
Risk	Process/ Time	-	-
Scenario	Process/ Time	-	-
Quality Attribute	Process/ Time	-	-
Product Value	Benefit	-	-
End User Satisfaction Level	Benefit	-	-
Priority of a Requirement	Priority	-	-
Non-functional Requirement	Process/ Time	-	-
-	-	Release	Process/ Time
-	-	Development Path	Process/ Time
-	-	Development Step	Process/ Time
-	-	Total Cost of a Development Path	Cost
-	-	Total Cost of a Development Step	Cost
Software product	Process/ Time	Software Product	Process/ Time

Table 11 RTDQM vs TDQM: First part of the Table — similar concepts, Second part of the Table — RTDQM specifics, Fourth part of the Table — CTDQM specifics, Fifth part of the Table — common concepts, *In Italics* — Concepts most commonly discussed among the RTD quantification approaches in our dataset

Summary of the comparison between RTD and CTD:

- Although some concepts are similar in both models, they can relate to different artifacts.
- RTD interest can encompass multiple stages in software development. Unlike CTD, it can impact both upstream and downstream activities in software development.
- Even though some concepts appeared to be specific to CTD, they were also found in RTD quantification approaches.
- Concepts specific to RTD emphasized the end user's involvement in RTD.

10 Implications to Researchers and Practitioners and Future Work

10.1 Implications to Researchers

A definition for RTD and a reference for RTD quantification

SMS results indicated that there is no commonly agreed definition for RTD and that there is no commonly agreed way to quantify RTD. There was no common reference point for understanding RTD quantification in the existing literature. Therefore, we formally defined RTD and developed a conceptual model to unify knowledge related to RTD quantification so that we could compare and analyze existing approaches based on the model. Our model also guides the development of new quantification approaches and helps understand what concepts may need to be supported by metrics.

The value of having our conceptual model is that it reveals aspects of RTD quantification that were not directly visible in the existing literature for both functional and non-functional requirements; one such aspect is the consequences of RTD in the form of constituents of RTD Interest. Another aspect is the costs and benefits associated with RTD.

There is still a lack of metrics for RTD quantification

There is still a lack of metrics for quantifying RTD. This became evident from analyzing the existing quantification approaches based on our model. This is discussed in Section 8.1. However, the model helps identify concepts related to RTD quantification and identify where metrics could support the quantification of those concepts. The model can guide the development of new quantification approaches. This use of the model is discussed in Section 8.2.

The comparison of the existing approaches revealed that the quantification of the consequences of RTD (i.e., RTD Interest) was discussed among most approaches. However, regardless of whether the approaches are categorized as discussing functional or non-functional aspects of requirements, all approaches have no metrics associated with RTD Interest and its constituents. Furthermore, most of the benefit concepts, the priority of a RTD Item, and RTD Interest probability also lack metrics. Investigating metrics that can support the quantification of these concepts is a potential future research direction.

The end-user is important for RTD management

Combining RTDQM with TDQM allowed a clear comparison of RTD with the TD types related to software code. The combined model shows the feedback loop involved with the user when capturing user needs and meeting the users' level of satisfaction for RTD, which differs from TD types related to the software code.

TD quantification can inform RTD quantification

Having discussed a comparison of code-related TD quantification and RTD quantification concepts in Section 9, the combined model enables one to develop new metrics to support the quantification of RTDQM concepts by being informed by the metrics that have been developed for code-related TD quantification concepts in the TD literature (and in the industry). In this paper, we did not perform a detailed comparison of metrics for CTD and RTD. This is a possible future work that can be done by using our model as a reference point.

Developing instruments for future research

The model can also be used to develop instruments for future research, for example, to develop surveys that investigate whether practitioners quantify RTD as discussed in the model. By doing so, researchers can investigate whether software practitioners quantify the costs and benefits associated with RTD captured in the model and whether they utilize metrics to support the quantification of the concepts.

10.2 Implications to Practitioners

When sub-optimal decisions concerning requirements are made inadvertently, the ability to identify them and rectify them early on is important. When sub-optimal decisions are made deliberately (in a prudent manner), for example, to deliver the most important set of features on time to market or to develop a prototype that helps gain user feedback, then such RTD must be monitored and rectified prior to facing larger problems. Furthermore, there may be situations where one set of requirements must be prioritized over another while being aware of the trade-offs. All these are instances where practitioners have to make informed decisions with respect to managing the RTD.

Practitioners may have particular needs regarding what they want to quantify when making informed decisions. They may want answers to particular questions such as the following when making their decisions for RTD management.

- How much RTD has accumulated in the project right now?
- What is the Interest we are paying for not fixing the problems with requirements?
- What is the benefit that can be gained by fixing problems with requirements right now rather than later?
- What is the impact on the project due to changes in the requirements (i.e., What is the impact of the implementation being out of sync with the requirements)?
- What are the consequences of RTD for a given set of sub-optimally prioritized requirements?

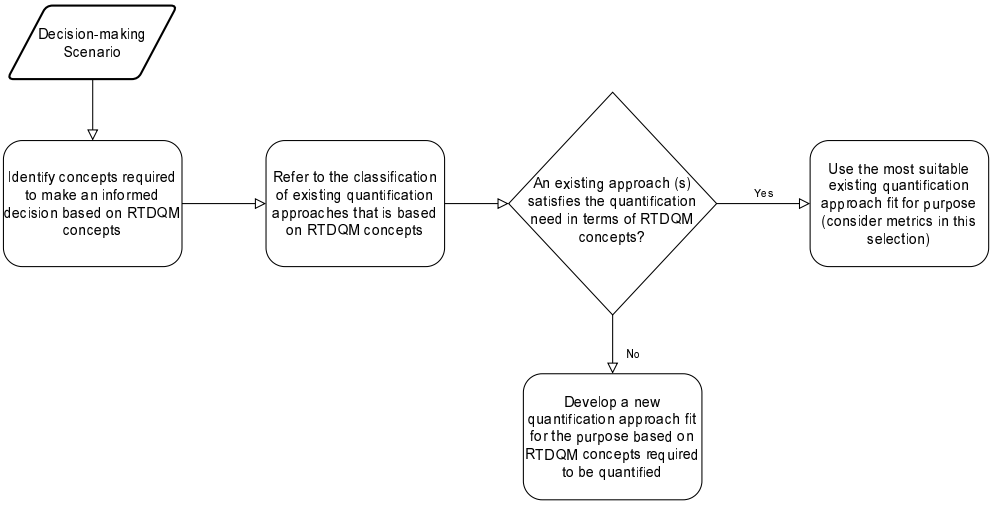

Fig. 10 Practitioners' Flow Chart for informed-decision making

Depending on their quantification needs, practitioners may want to decide what existing quantification approaches fit their purpose. For this, they need to ask two questions: *What concepts should be part of the quantification? and Is there a good existing quantification approach that supports making an informed decision in the given scenario?* Figure 10 shows how practitioners may go about answering these questions.

See Figure 10; in a given decision-making scenario (e.g., the decision to take on or not take on RTD with a sub-optimal prioritization of requirements), a practitioner identifies the concepts that may help make an informed decision based on RTDQM concepts. Then, they can refer to the classifications of existing quantification approaches presented in Section 8.1 in this paper (see Figure 7).

Our classification of the existing quantification approaches discussed in Section 8.1 shows what concepts are discussed in the existing RTD quantification approaches and what approaches may have metrics to support the quantification of the RTDQM concepts (see Figure 7). Practitioners could use this classification to select an approach that helps satisfy their quantification needs.

If there are existing quantification approaches that discuss the relevant concepts in our conceptual model (i.e., RTDQM concepts) and provide metrics (or some form of measurement) for their quantification, practitioners could select an approach that best fits their purpose based on that. If an existing approach does not support the quantification, practitioners may want to develop new quantification approaches and supporting metrics to satisfy their quantification needs. Section 8.2 discusses an example related to a RTD management decision-making scenario (See 8.2.3 — selecting an existing approach and Section 8.2.3 — identifying the need to develop a quantification approach).

However, the RTD quantification approaches that were identified in the literature discuss different aspects that must be quantified, and some of them do not provide adequate support to quantify the RTDQM concepts they discuss. This lack of consensus

was initially indicated by the SMS results and became more evident from the analysis and classification of the existing quantification approaches discussed in Section 8.1. Therefore, we encourage the development of new quantification approaches and metrics to support the quantification of RTDQM concepts that could eventually support informed decision-making for RTD management.

10.3 Future Work

This work laid the theoretical foundation for quantifying RTD, proposing how researchers and practitioners can go about utilizing existing quantification approaches and developing new quantification approaches to support RTD quantification.

Our future work employs a survey with practitioners from the industry to learn whether practitioners are currently aware of RTD quantification and to gather their perceptions on how the quantification of RTD could be better supported.

In the survey, we inquire whether practitioners formally or informally quantify RTD, and if so, how they quantify it and what their perceptions are about quantifying the costs, benefits, and consequences of RTD so that the gaps in these can be addressed by research. Our conceptual model, RTDQM, which we presented in this paper, guided the development of the survey instrument.

11 Threats to Validity

We discuss threats to the validity of our SMS and model development based on the guidelines provided by Petersen et al. [43].

11.1 Descriptive Validity

Descriptive Validity refers to how a study describes its observations accurately and objectively. This threat applies to our SMS and the development of the conceptual model.

To reduce the threat to the SMS, we recorded the data we extracted from the papers in a tabular format. Demographic data, such as the paper title, publication year, venue, author names, and the number of citations, was automatically recorded from SCOPUS. The first author extracted and analyzed the rest of the data objectively, based on the themes described in Section 3 (SMS Methodology). The recorded data was discussed among all authors before their analysis to determine how they would be analyzed and reported in this paper.

For the model development, data was extracted and analyzed using the systematic coding process adopting the Thematic Analysis approach [23] described in Section 6, while Figure 2 provides an overview Figure 3 provides an example. The concepts extracted from the papers were mapped to the model concepts using a matrix format (*TDQM Approach Comparison Matrix*, adopted from Perera et al.'s work [5]). At least two other authors reviewed the data extraction and analysis in iterative meetings. Concerns were discussed and resolved during the iterative consensus meetings. All authors agreed on the final set of model concepts before they were included in the model.

11.2 Theoretical Validity

Theoretical Validity refers to the ability to capture what we intend to capture through our study. This applies to identifying primary studies in the SMS and data extraction in both the SMS and model development.

11.2.1 Identification of Primary Studies

We improved our search query to be less restrictive by removing specific phrases and using the 'AND' operation to combine keywords (e.g., "technical debt" AND "software requirements" instead of "Technical Debt in Software Requirements"). The list of quantification-related keywords was removed from the query string to retrieve papers that may have been filtered out otherwise. The search query was applied 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.

Additionally, we conducted Backward Reference Snowballing to complement the results found via the search query. The research questions, inclusion/exclusion criteria, and the selection of studies were discussed among all authors. At least two other authors evaluated the results at each stage of the screening. Therefore, this threat has been sufficiently mitigated.

11.2.2 Data Extraction and Analysis

The first author performed the data extraction and analysis of the primary studies in the SMS and the data extraction analysis for the model development.

Themes described in Section 3 were used for the data extraction in the SMS. The extracted data was reviewed by the other authors prior to analysis. The authors discussed and agreed on how the data will be analyzed and reported in this paper.

For the model development, data was first extracted using the systematic coding process adopting the Thematic Analysis approach [23] as described in Section 6. Mappings between initial codes and abstract concepts were performed using the TDQM Approach Comparison Matrix developed by Perera et al. [5], an example is discussed with reference to Ernst at. al. (Primary Study P7) — see Figure 3 and Section 6.1). The first author performed the mappings. At least two other authors verified the mappings. The resulting model concepts and relationships were discussed and agreed upon between all authors prior to including them in the model. Disagreements that were raised were resolved during meetings. The interpretations of the results and the conclusions were also discussed among the authors before they were reported in this paper.

11.3 Generalizability

Generalizability refers to the external and internal generalizability of our study. SMSs typically follow a common process. We followed the guidelines by Petersen et al. [43]. Therefore, the threat to internal generalizability is mitigated. However, study goals may differ for different SMSs. Therefore, external generalizability cannot be discussed with respect to SMSs.

We do not claim that our model is complete and generalizable to all contexts and scenarios in software development. We cannot make this claim without validating our model in an industrial setting. However, our model captures the concepts and relationships sufficient to model RTD quantification and to serve as a common reference point to compare and analyze existing quantification approaches. We evaluated this by applying our model to the approaches found in our SMS (see Section 8.1).

11.4 Interpretive Validity

Interpretive validity refers to conclusion validity. The conclusions should be reasonable given the data [43]. The first author primarily made the conclusions of our study, and this might be subject to researcher bias. However, we discussed the results and conclusions among all authors in multiple iterative meetings prior to reporting them in this paper. Therefore, this threat is sufficiently mitigated.

11.5 Repeatability

The detailed reporting of a research process preserves the repeatability of the study. We have reported the SMS process in Section 1 and the overview in Figure 1. The process we followed applied the guidelines by Petersen et al. [43], a commonly followed process for SMSs by the research community.

Our systematic coding to developing our model adopting the Thematic Analysis approach by Braun and Clarke [23] is described in Section 6, an overview of the methodology and an example of the systematic coding are provided in Figures 2 and 3, respectively.

Furthermore, we provide a Replication Package[1] as supplementary material.

12 Conclusion

We formally defined Requirements Technical Debt (RTD) and developed a conceptual model for RTD quantification, the *Requirements Technical Debt Quantification Model (RTDQM)*, as a theoretical foundation to support the understanding of RTD quantification. Our initial work was reported in our conference paper [3]. This paper extends our initial work, improving the SMS and the model. We report on results from all 18 primary studies and the findings from the improvements made to the model.

RTDQM was improved to model the functional and non-functional aspects of requirements for RTD quantification based on 286 concepts extracted from the literature found in our SMS. The result was combined with prior work modeling the quantification of code-related TD to have a more complete model of both requirements TD and TD related to the software code.

The key observation from our work is that although RTD is similar to code-related TD in many aspects, it has its own components. RTD can be incurred independently, regardless of whether there is code-related TD in a software project. RTD can impact both upstream and downstream activities in software development. For example, RTD Interest can be incurred during Requirements Engineering (RE) and also

[1] Replication Package: https://doi.org/10.5281/zenodo.10900222

during Design and Implementation. The impact on downstream activities can also be cascading from the upstream activities. Unlike code-related TD, RTD has a feedback loop involving the User to precisely capture their needs and meet their level of satisfaction. This became more evident when the code-related and requirements TD quantification models were combined.

RTDQM unifies the knowledge of RTD quantification gained from the existing literature. It can serve as a reference point to compare existing approaches for RTD quantification and to develop new approaches, metrics, and tools to support informed decision-making based on the model concepts and their relationships. The model's utility was evaluated by applying it to compare and analyze existing quantification approaches found through our SMS and by illustrating, via an example case study, how the model can help practitioners select an existing approach or identify the need to develop new approaches to support their particular quantification needs. The model can guide the development of instruments for empirical research. Our future work employs a survey developed based on the model to gather software practitioners' perceptions of RTD quantification.

Supplementary information. Replication Package: https://doi.org/10.5281/zenodo.10900222

Acknowledgments. This research is funded by the New Zealand Ministry of Business, Innovation, and Employment via The Science for Technological Innovation (SfTI) National Science Challenge Veracity Technology Spearhead.

Appendix A SMS Primary Studies

	Approach Description
P1	Applies the *real options theory* to quantify requirements decisions in the form of their Net Present Value (NPV) by considering uncertainty in requirements selection, schedule and final cost. A Positive number for NPV indicates that the current requirements specification does not take on RTD.
P2	Extends Ernst et al.'s RTD definition to include upstream activities involving the elicitation of requirements and their translation into specifications. Defines how to identify, quantify, and payback RTD for RTD types, 0, 1, 2.
P3	A *tool-based approach* to prevent requirements documentation TD during RE. Integrates the SRS into the IDE enabling real-time traces between requirements and code.
P4	Investigates the causes that incur RTD and actions that can minimize and or avoid them in the context of agile software development.
P5	Focuses on RTD incurred for a given feature as the extent to which that feature is not implemented. Relies on an uncertainty measure to assess RTD accrued for a specific feature (i.e., to assess the extent to which that feature is not entirely supported by the system). If the INCIDENCE value is above a particular threshold for the top claim of the case, then there is sufficient certainty that the feature is supported by the system.
P6	Performs a live study surveying RE experts to gain further insights on the issues taking place at this stage and how they fit in their definition of RTD Type 1: Requirement Smells, an indicator for a quality violation of a requirements artifact. Aims to understand and compare the perceived harmfulness of requirement smells from a theoretical and practical perspective.
P7	Introduces a *tool* that helps compare one implementation to new proposed implementations to reduce RTD.
P8	Defines the concept of TD for managing compliance requirements exploring its link to compliance goals and obstacles. Requirements are represented in the form of compliance goals that range from high level business objectives to well-defined compliance properties. These compliance requirements can be related to quality attributes such as information security. Obstacles represent undesired properties that prevent satisfying the goals. Cost required to resolve an obstacle is the TD Principal, the extra costs that will be incurred in case the obstacle is not resolved, is the TD Interest.
P9	The goal of the conceptual model introduced in this paper is to support optimal QR documentation in ASD. The model provides a representation and explanation of the factors affecting QR documentation and identifies mitigation strategies to overcome the issues introduced due to those factors. The model is based on 3 case studies. Practitioners can analyze for example, how time constraints or QR awareness affects documentation, see potential issues that may arise from them, and utilize strategies suggested by the model to address these issues.
P10	Proposes a RED theory that aligns concepts from TD research but emphasizes the specific nature of requirements engineering. The theory is based on interview data. The authors put together a list of causes and consequences and prevention and repayment practices for RED based on literature.

P11	'Sustainability' can be viewed as a concern that can cross-cut both functional and non-functional requirements. This paper proposes an economics-driven architecture evaluation method which extends CBAM and integrates principals of MPT to evaluate software architecture design decisions for sustainability. The approach aims to identify architecture design decisions which minimize costs, reduce risk and maximize value on sustainability dimensions of interest including – environmental, social, economic, technical, and individual– and calculates the requirements debt related to these dimensions. This approach models technical debt as a function of requirements trade-offs (i.e. QA response trade-offs for scenarios of interest).
P12	The authors suggest that, for fulfillment of Quality Requirements (QRs), the consequences should be evaluated and adequate resources must be allocated to develop a solution that meets the requirements. They combine existing techniques such as, Quality Attribute Workshops (QAW) and Impact Mapping to create a lightweight, iterative and effective quality requirement elicitation process.
P13	Aims to present a process for managing UTD supporting the debts identification, validation, rating, prioritization, estimation and monitoring. The process involves actors and tasks to be performed by the actors, for example, UTD analysts will focus on identifying usability issues and following up on their correction, development team will act to validate problems in UTD and prioritize, estimate, and pay the UTD
P14	Investigates the impacts of agile requirements documentation debt on software projects. The findings indicated an extra maintenance effort of about 47 percent of the total effort estimated for developing the project and an extra cost of about 48 percent of the initial cost of the development phase in their retrospective case study.
P15	Proposes a market driven systematic approach to supplement the selection of requirements, which accounts for uncertainty and incomplete knowledge in the real world. Utilizes portfolio-based reasoning to make the connection to market value explicit.
P16	Investigates the state of the practice of R2DD, understanding causes, effects, and practices and practice avoidance reasons (PARs) for debt prevention and repayment and organizes those practices into a conceptual map. The authors emphasize that it is important to discuss the management of TD in the context of requirements engineering activities because they are inherently complex and impact several software development phases (e.g., coding, test planning).
P17	Presents CBAM (Cost Benefit Analysis Method), in which costs and benefits are traded off with system quality attributes. CBAM builds upon the Architecture Tradeoff Analysis Method (ATAM) to model the costs and benefits of architectural decisions and to provide means of optimizing such deicisons. CBAM helps in the elicitation and documentation of costs, benefits and uncertainity and gives the stakeholders a rational decision-making process.
P18	Introduces a cost-value approach that prioritizes requirements according to their relative value (in terms of customer satisfaction) and costs (in terms of implementation costs). Defines 'quality' as, candidate requirement's potential contribution to customer satisfaction with the resulting system. Defines 'cost' as, cost of successfully implementing the candidate requirement.

Table A1: Short descriptions of the RTDM Approaches

References

[1] Bonfim, V.D., Benitti, F.B.V.: Requirements debt: causes, consequences, and mitigating practices. In: SEKE, pp. 13–18 (2022)

[2] Ernst, N.A.: On the role of requirements in understanding and managing technical debt. In: 2012 Third International Workshop on Managing Technical Debt, pp. 61–64 (2012). IEEE

[3] Perera, J., Tempero, E., Tu, Y.-C., Blincoe, K.: Quantifying requirements technical debt: A systematic mapping study and a conceptual model. In: 2023 IEEE 31st International Requirements Engineering Conference (RE), pp. 123–133 (2023). https://doi.org/10.1109/RE57278.2023.00021

[4] Abad, Z.S.H., Ruhe, G.: Using real options to manage technical debt in requirements engineering. In: 2015 IEEE 23rd International Requirements Engineering Conference, pp. 230–235 (2015). IEEE

[5] Perera, J., Tempero, E., Tu, Y.-C., Blincoe, K.: Quantifying technical debt: A systematic mapping study and a conceptual model. arXiv preprint arXiv:2303.06535 (2023) https://doi.org/10.48550/arXiv.2303.06535

[6] Guarino, N., Guizzardi, G., Mylopoulos, J.: On the philosophical foundations of conceptual models. Information Modelling and Knowledge Bases **31**(321), 1 (2020)

[7] Delcambre, L.M., Liddle, S.W., Pastor, O., Storey, V.C.: A reference framework for conceptual modeling. In: Conceptual Modeling: 37th International Conference, ER 2018, Xi'an, China, October 22–25, 2018, Proceedings 37, pp. 27–42 (2018). Springer

[8] Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. Journal of Systems and Software **101**, 193–220 (2015)

[9] Cunningham, W.: The WyCash portfolio management system. Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA **Part F1296**(October), 29–30 (1992) https://doi.org/10.1145/157709.157715

[10] Rios, N., Mendonça Neto, M.G., Spínola, R.O.: A tertiary study on technical debt: Types, management strategies, research trends, and base information for practitioners. Information and Software Technology **102**, 117–145 (2018)

[11] Avgeriou, P., Kruchten, P., Ozkaya, I., Seaman, C.: Managing technical debt in software engineering (dagstuhl seminar 16162). In: Dagstuhl Reports, vol. 6 (2016)

[12] Melo, A., Fagundes, R., Lenarduzzi, V., Santos, W.B.: Identification and measurement of requirements technical debt in software development: A systematic literature review. Journal of Systems and Software, 111483 (2022)

[13] Lenarduzzi, V., Fucci, D.: Towards a holistic definition of requirements debt. In: 2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–5 (2019). IEEE

[14] Frattini, J., Fucci, D., Mendez, D., Spínola, R., Mandić, V., Taušan, N., Ahmad, M.O., Gonzalez-Huerta, J.: An initial theory to understand and manage requirements engineering debt in practice. Information and Software Technology **159**, 107201 (2023) https://doi.org/10.1016/j.infsof.2023.107201

[15] Nugroho, A., Visser, J., Kuipers, T.: An empirical model of technical debt and interest. In: Proceedings of the 2nd Workshop on Managing Technical Debt, pp. 1–8 (2011)

[16] Alves, N.S., Mendes, T.S., Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C.: Identification and management of technical debt: A systematic mapping study. Information and Software Technology **70**, 100–121 (2016)

[17] Behutiye, W.N., Rodríguez, P., Oivo, M., Tosun, A.: Analyzing the concept of technical debt in the context of agile software development: A systematic literature review. Information and Software Technology **82**, 139–158 (2017)

[18] Kitchenham, B., Pretorius, R., Budgen, D., Brereton, O.P., Turner, M., Niazi, M., Linkman, S.: Systematic literature reviews in software engineering–a tertiary study. Information and software technology **52**(8), 792–805 (2010)

[19] Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M.: Systematic mapping studies in software engineering. In: 12th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10 (2008)

[20] Brereton, P., Kitchenham, B.A., Budgen, D., Turner, M., Khalil, M.: Lessons from applying the systematic literature review process within the software engineering domain. Journal of systems and software **80**(4), 571–583 (2007)

[21] Cavacini, A.: What is the best database for computer science journal articles? Scientometrics **102**(3), 2059–2071 (2015)

[22] Wohlin, C.: 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, pp. 1–10 (2014)

[23] Braun, V., Clarke, V.: Using thematic analysis in psychology. Qualitative research in psychology **3**(2), 77–101 (2006)

[24] Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Tsiridis, N.: Integrating traceability within the ide to prevent requirements documentation debt. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications, pp. 421–428 (2018). IEEE

[25] Belle, A.B., Lethbridge, T.C., Kpodjedo, S., Adesina, O.O., Garzón, M.A.: A novel approach to measure confidence and uncertainty in assurance cases. In: 2019 IEEE 27th International Requirements Engineering Conference Workshops, pp. 24–33 (2019). IEEE

[26] Lenarduzzi, V., Fucci, D., Mendéz, D.: On the perceived harmfulness of requirement smells: An empirical study. In: Joint 26th International Conference on Requirements Engineering: Foundation for Software Quality Workshops, Doctoral Symposium, Live Studies Track, and Poster Track, Pisa; Italy, vol. 2584 (2020). CEUR-WS

[27] Ojameruaye, B., Bahsoon, R.: Systematic elaboration of compliance requirements using compliance debt and portfolio theory. In: Requirements Engineering: Foundation for Software Quality: 20th International Working Conference, REFSQ 2014, Essen, Germany, April 7-10, 2014. Proceedings 20, pp. 152–167 (2014). Springer

[28] Behutiye, W., Rodríguez, P., Oivo, M., Aaramaa, S., Partanen, J., Abhervé, A.: Towards optimal quality requirement documentation in agile software development: A multiple case study. Journal of Systems and Software **183**, 111112 (2022)

[29] Frattini, J., Fucci, D., Mendez, D., Spinola, R., Mandić, V., Taušan, N., Ahmad, M.O., Gonzalez-Huerta, J.: An initial theory to understand and manage requirements engineering debt in practice. Information and Software Technology **159**, 107201 (2023)

[30] Mohagheghi, P., Aparicio, M.E.: An industry experience report on managing product quality requirements in a large organization. Information and Software Technology **88**, 96–109 (2017)

[31] Ojameruaye, B., Bahsoon, R., Duboc, L.: Sustainability debt: A portfolio-based approach for evaluating sustainability requirements in architectures. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 543–552 (2016)

[32] Costa, A.F.F., Marques, A.B.D.S., Santos, I.S., Andrade, R.M.D.C.: Towards a process to manage usability technical debts. In: Proceedings of the XXXVI Brazilian Symposium on Software Engineering, pp. 241–246 (2022)

[33] Mendes, T.S., F. Farias, M.A., Mendonça, M., Soares, H.F., Kalinowski, M.,

Spínola, R.O.: Impacts of agile requirements documentation debt on software projects: a retrospective study. In: Proceedings of the 31st Annual ACM Symposium on Applied Computing, pp. 1290–1295 (2016)

[34] Sivzattian, S., Nuseibeh, B.: Linking the selection of requirements to market value: A portfolio-based approach. In: Proceedings of 7th International Workshop on Requirements Engineering: Foundation for Software Quality (REFSQ 2001) (2001)

[35] Barbosa, L., Freire, S., Rios, N., Ramač, R., Taušan, N., Pérez, B., Castellanos, C., Correal, D., Pacheco, A., López, G., et al.: Organizing the td management landscape for requirements and requirements documentation debt. UMBC Faculty Collection (2022)

[36] Kazman, R., Asundi, J., Klein, M.: Quantifying the costs and benefits of architectural decisions. In: Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001, pp. 297–306 (2001). IEEE

[37] Karlsson, J., Ryan, K.: A cost-value approach for prioritizing requirements. IEEE software **14**(5), 67–74 (1997)

[38] Charalampidou, S., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: Assessing code smell interest probability: a case study. In: Proceedings of the XP2017 Scientific Workshops, pp. 1–8 (2017)

[39] Junior, H.J., Travassos, G.H.: Consolidating a common perspective on technical debt and its management through a tertiary study. Information and Software Technology, 106964 (2022)

[40] Sommerville, I.: Sommerville: Software Engineering. Pearson (2011)

[41] Ojameruaye, B., Bahsoon, R.: Sustainability archdebts: An economics-driven approach for evaluating sustainable requirements. Software Sustainability, 369–398 (2021)

[42] Bass, L., Clements, P., Kazman, R.: Software architecture in practice. Addison-Wesley Professional (2007)

[43] Petersen, K., Vakkalanka, S., Kuzniarz, L.: Guidelines for conducting systematic mapping studies in software engineering: An update. Information and software technology **64**, 1–18 (2015)