Quantifying Requirements Technical Debt: A Systematic Mapping Study and a Conceptual Model

Judith Perera, Ewan Tempero, Yu-Cheng Tu, Kelly Blincoe
The University of Auckland, New Zealand
jper120@aucklanduni.ac.nz, {e.tempero, yu-cheng.tu, k.blincoe}@auckland.ac.nz

***Abstract*—Requirements Technical Debt (RTD) is a research area where the Technical Debt (TD) metaphor is used 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 model the quantification of RTD. Our work is grounded in the literature found via a Systematic Mapping Study (SMS) and informed by prior work modeling the quantification of TD for software code-related TD types. This paper reports on the SMS and the development of our model, the *RTD Quantification Model (RTDQM)*. The key observation from our work is that, although RTD is similar in most aspects to TD in software code, it also has its own components. Requirement artifacts have a feedback loop involving the User to precisely capture User Needs. RTD Interest (i.e., additional costs due to sub-optimal decisions concerning Requirements) can incur during both Requirements Engineering and Implementation activities. Furthermore, RTD can incur regardless of the presence of software code-related TD. Similar to benefits accrued by refactoring software code, rectifying RTD can also accrue benefits.**

***Index Terms*—Requirements Technical Debt Quantification, Systematic Mapping, Conceptual Model**

I. Introduction

Software Requirements play a crucial role in the development of a software product — gathering requirements right and delivering the product that offers the best value to the end-users is essential for the success of a software product [7], [14]. However, stakeholders involved in Requirements Engineering (RE) activities (e.g., business analysts, requirements engineers) and developers implementing the requirements into features of a software product can make sub-optimal decisions either deliberately or inadvertently during RE and implementation activities. Requirements Technical Debt (RTD) captures the consequences (in terms of costs) of such sub-optimal decisions made concerning requirements. To manage RTD, it must be quantified [1]. However, there is little evidence in the literature on how RTD can be quantified [1], [3]. Therefore, as a means to better understand the quantification of RTD, we establish a theoretical foundation for this problem by focusing on RTD and its quantification in this paper.

We first conducted a Systematic Mapping Study (SMS) to identify what the existing literature proposes on RTD and its quantification. We then developed a conceptual model, the ***RTD Quantification Model (RTDQM)*, the main contribution of this paper**, to capture the concepts related to RTD quantification and the relationships between them. The model concepts and relationships are grounded in the literature identified through the SMS and are additionally informed by prior work of Perera et al. [25] who developed a model[1] for the quantification of software code-related types of TD such as code, design, and architectural TD[2] (discussed in Section II). Our study answers the following Research Questions:

- ***RQ1: What can we learn about quantifying RTD from approaches proposed in RTD literature? — answered via the SMS***
- ***RQ2: How can we model the quantification of RTD? — answered via the development of the conceptual model***

Findings for RQ1 revealed that different authors discuss different definitions of RTD and there is no commonly agreed definition nor a commonly agreed way to quantify RTD, which led us to formally define RTD and then model RTD quantification via RQ2. The model's development enabled a deeper understanding of RTD and its quantification. The key observation is that, although RTD is similar to CTD in some aspects, it also has its own components. RTD Items can be introduced during RE or Implementation activities regardless of the presence of CTD. Different from CTD, RTD has a feedback loop involving the User. RTD Interest can incur extra costs associated with RE as well as Implementation. Similar to benefits accrued by refactoring code, rectifying RTD can also accrue benefits.

The paper begins by providing background and discussing related work in Section II, then reports on the SMS Methodology and Results in Sections III and IV. Section V describes the methodology for developing RTDQM and describes RTDQM in Section VI. Section VII discusses what we learned from the development of the model, threats to validity, implications to researchers and practitioners, and future work. The paper is concluded in Section VIII.

II. Background and Related Work

A. Technical Debt (TD), TD Management and Quantification

Technical Debt (TD) captures the consequences of sub-optimal decisions made during the development of a software product [13], [24]. These sub-optimal decisions could be made throughout the software development lifecycle, from requirements engineering to product maintenance [27]. Li et al. [21], identified that TD Management (TDM) is important in the implementation and maintenance of software. According

[1]The Technical Debt Quantification Model (TDQM) [25].
[2]Code-related types of TD will be referred to as CTD here onward.

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. [25], identified during their SMS that quantification approaches proposed in the literature lack consensus. Therefore, the authors developed a Conceptual Model based on findings from the mapping study and the work of Avgeriou et al. [4] to model the quantification of CTD. 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 [25]. 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.

B. RTD, RTD Management (RTDM) and Quantification

Compared to CTD, RTD is a relatively new area with less literature on the topic [1], [22]. Furthermore, there is no consensus yet if RTD can be considered a type of TD [19]. 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 et al. [14], Lenarduzzi et al. [19], and Frattini et al. [15]. Below, we discuss the most relevant related work in RTD research.

Ernst et. al, 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* [14]. According to Ernst et. al, RTD describes tradeoffs on 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 [14]. 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 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 [19]. 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 (this study is discussed further in Section IV).

Frattini et. al [15], emphasizes the inadequacy of exploring the debt metaphor in the context of RE. As a result they develop an initial analytical theory as a first 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 [15]. 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.

C. 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. [22]. 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., [23], is not focused on RTD. Compared to their study, our SMS focuses on studies discussing RTD 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 (ours includes studies from 2020-2022 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.

D. Definitions of RTD from Secondary Studies

We examined RTD definitions given in previous secondary studies where RTD was mentioned. Li et al. [21] and Melo et al. use the early definition given by Ernst et al. [14]. Alves et al. [2] refer to a similar definition; they define RTD as trade-offs made by the development team with respect to what requirements 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. [5] 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 III.

III. SMS — METHODOLOGY

We followed recommendations given by Kitchenham et al. [18] and Petersen et al. [26] to conduct the SMS which answers the following research question:

- ***RQ1: What can we learn about quantifying RTD from approaches proposed in RTD literature?***

A. Search Strategy

We used SCOPUS as the primary database to build our search query. The final search query (See Figure 1) was then tailored to other databases according to their functionality. The final query contained the following search terms and synonyms: *Requirements Technical Debt, quantify, measure, forecast, predict, assess, estimate, calculate, amount, value, impact, principal, interest, metric, time, cost*." Additionally, we used the phrase 'Technical Debt in User Stories' to capture papers that did not use the phrase 'Requirements' but still referred to RTD. We avoided using the term 'Technical Debt' on its own, to avoid articles that did not focus on RTD but on other TD types. We applied the query to the title, abstract, keywords, and full text to increase the possibility of finding all relevant articles. Digital databases IEEEXplore, ACM, and Science Direct were selected based on recommendations by Brereton et al. [9]. SCOPUS was selected following the recommendations by Cavacini [10]. Duplicates were removed before screening articles for inclusion.

(ALL ("Requirements Debt") OR ALL ("Requirements Technical Debt") OR ALL ("Technical Debt in Requirements") OR ALL ("Technical Debt in Requirements Engineering") OR ALL ("Technical Debt in Software Requirements") OR ALL ("Technical Debt in User Stories")) AND (quantif* OR measur* OR forecast* OR predict* OR assess* OR estimate* OR calculat* OR impact* OR amount OR valu* OR principal OR interest* OR metric* OR time OR cost*) AND (LIMIT-TO (DOCTYPE , "cp") OR LIMIT-TO (DOCTYPE , "ar")) AND (LIMIT-TO (LANGUAGE , "English"))

Fig. 1. Final Search Query for SCOPUS

Inclusion	I1 Discusses the quantification of RTD or provides some direction 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
	E4 Secondary or Tertiery Studies
	E5 Not RTD
	E6 Not peer reviewed

TABLE I
INCLUSION AND EXCLUSION CRITERIA

B. Article Screening and Selection

Article screening and selection were performed following the *adaptive depth reading approach* suggested in Petersen et al. [26] starting from the title and then continuing through the abstract, conclusion, and at last, reading the full text. Additionally, we applied the inclusion/exclusion criteria listed in Table I. The first author screened the articles and whenever doubt was encountered, they were discussed with the other authors during meetings.

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, the full text was inaccessible and did not discuss RTD. Only peer-reviewed conference and journal articles were included as they are considered high quality.

C. Forward and Backward Reference Snowballing

Forward and Backward reference snowballing [28] was performed on the final set of articles to avoid the possibility of missing relevant articles. Backward snowballing was guided by the related work sections in the articles. Forward snowballing was conducted in SCOPUS, the most comprehensive and largest database among the databases we queried.

D. Data Extraction, Analysis and Synthesis

The first author performed data extraction, analysis and synthesis following recommendations given by Petersen et al. [26] and Braun and Clarke [8] 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 [8] was followed for data analysis and synthesis. It is an effective method for identifying, analyzing, and reporting patterns and themes within data [8]. The high level themes resulted 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 IV.

IV. SMS — RESULTS (RQ1)

Below we report the results derived from 7 articles included out of 87 articles obtained from the digital databases.

A. Demographics

Publications resulting from the SMS range from 2012 - 2022. The publication venues vary from different conferences to workshops. The oldest (2012) and most cited (27 in SCOPUS) publication is by Ernst et al. The next highest cited article is by Abad et. al, published in 2015 (See Table II).

B. RTD Definitions

The earliest definition of RTD was given by Ernst et al. [14] in Table II), which was discussed earlier in Section II. Lenarduzzi et al. [19], introduced a more comprehensive definition including 3 types of RTD incurred during upstream

	Title	Pub Year	Authors, Citation	Venue	Citations
P1	Using real options to manage Technical Debt in Requirements Engineering	2015	Abad et. al, [1]	RE	18
P2	Towards a Holistic Definition of Requirements Debt	2019	Lenarduzzi et. al, [19]	ESEM	7
P3	Integrating Traceability Within the IDE to Prevent Requirements Documentation Debt	2018	Charalampidou et. al, [12]	SEAA	2
P4	Requirements debt: causes, consequences, and mitigating practices	2022	Bonfim et. al, [7]	SEKE	0
P5	A novel approach to measure confidence and uncertainty in assurance cases	2019	Belle et. al, [6]	REW	2
P6	On the perceived harmfulness of requirement smells: An empirical study	2020	Lenarduzzi et. al, [20]	CEUR	0
P7	On the role of requirements in understanding and managing technical debt	2012	Ernst et. al, [14]	MTD	27

TABLE II
DEMOGRAPHICS: PUBLICATION YEAR, AUTHORS, CITATION AND VENUE — NUM OF CITATIONS FROM SCOPUS

activities such as identification and formalization of requirements apart from the debt incurred during the implementation of requirements (See Table III). While they introduce two other types of RTD, 0, and 1, their type 2 is similar to Ernst et al,'s (P7) initial definition. Abad et al. [1], 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., [11] 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., [7] (P4) define RTD as failures in SRS, which again aligns with type 2 in P2 (and P7). Belle et al. [6], define RTD as poor or partial implementation of requirements into features, this is also similar to type 2 in P2 (and P7). Therefore, we see that most of these studies use variations of the initial definition provided by Ernst et al. (P7) while some definitions refer to type 1 in P2. 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., is yet missing in P2's definition. Based on our findings, we synthesize the following definition for RTD: ***"Requirements TD 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, formalization, and implementation of requirements.***"

	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 loped 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.

TABLE III
RTD DEFINITIONS

C. Concepts related to RTD Quantification

We identified 57 concepts related to RTD quantification from the primary studies. Based on Pererea et al. [25] we categorized them into categories: process or time, cost, benefit, and probability which contained 32, 16, 1, 2 concepts, respectively. The list of concepts can be found in our Replication Package[3], we do not report them here due to space limits. Section VI and Table VIII describe the abstracted set of concepts that went into the conceptual model developed for RQ2.

D. Metrics, Tools and supported RTDM Activities

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

[3]Replication Package: doi.org/10.5281/zenodo.7563045

within an agile context. Although P3 and P4 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 Section VI, detailed mappings in Replication Package[3]).

	Approach Description	**Supported RTDM Activities**
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.	Quantification
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.	Identification, Quantification, Repayment
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.	Monitoring, Prevention
P4	Investigates the causes that incur RTD and actions that can minimize and or avoid them in the context of agile software development.	Prevention, Reduction
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.	Quantification
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.	Reduction/ Prevention, Quantification
P7	Introduces a *tool* that helps compare one implementation to new proposed implementations to reduce RTD.	Prioritization, Quantification

TABLE IV
RTDM APPROACHES AND SUPPORTED RTDM ACTIVITIES

E. Proposed RTDM Strategies

Table VI shows strategies to 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.

	Metrics
P1	Initial value of selected requirements in terms of the market payoff in market-driven or the product value in product-driven software development projects, 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, Cost of exercising the option (development cost), Present value of each node, Risk-adjusted probability, Current interest rate for a set of selected requirements over a specific time period, 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, Num. of Evidence Items (NOI), thresholds for INCIDENCE, extent to which experts estimate that a child node contributes to the proof of its parent node
P7	Distance, Interest or cost of neglecting the debt

TABLE V
RTD METRICS

	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

TABLE VI
RTDM STRATEGIES

F. RTD Causes, Indicators and Consequences of RTD

Similar to any TD type, RTD can be incurred or caused intentionally or unintentionally. Inefficiencies in identifying Requirements (P2), inefficiencies in the Requirements specification or Requirement Smells (P2, P3), and sub-optimal implementation of Requirements (P2, P3) can happen due to negligence or deliberate choice to not do so. If a system does not satisfy one or more non-functional requirements, this could also be a cause of RTD (P5). See Table VII, causes and indicators. P3 and P4 describe the consequences

	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
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

TABLE VII
CAUSES AND INDICATORS OF RTD

of RTD. 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, communication with customers on bug-fixing progress being hindered 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.

G. Challenges associated with RTD

Multiple challenges in the field of RE associated with RTD were derived from the studies. Common themes were: 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 in Requirements (P1, P7) — e.g., uncertainties about customer (or market) requirements, project context and environment, and the feasibility, cost, and duration of developing each requirement [1], and difficulties in monitoring and traceability of Requirements (P3, P7) — 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 [11], and only a few teams usually track Requirements [14]. Our study contributes to some of these challenges by establishing a formal definition for RTD as well as conceptualizing RTD quantification.

V. MODELLING RTD QUANTIFICATION — METHODOLOGY

Perera et al. [25], developed a model (TDQM[1]), for the quantification of TD types related to software code such as Code, Design and Architectural debt (CTD[2]). Observing the impact of TD introduced by non-code related artifacts such as Requirements can have on subsequent downstream software development activities motivated us to develop a model to conceptualize the quantification of RTD. The development of the conceptual model, *RTD Quantification Model (RTDQM)* 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 are the relationships that can be identified among those concepts?***

RTDQM (Figure 2) was developed in part by examining what constitutes RTD quantification informed by its code-related counterpart in Perera et al.'s work and in part by examining the literature captured via our SMS. Perera et al.,'s work has also been informed by literature captured via a SMS and by past models of TD, e.g., Avgeriou et al.'s work [4].

We examined primary studies in our SMS to extract various concepts related to RTD quantification. As discussed in Section IV, 57 concepts were initially extracted from the papers which were then abstracted into 14 concepts that went into the model for RTD quantification after multiple iterations of grouping together related concepts into themes and examining them with reference to the TDQM concepts and their themes — process or time, cost, benefit, and probability. These themes were recurrent among the concepts extracted in our study for RTD quantification as well. Perera et al.'s TDQM and their *Approach Comparison Matrix*, developed for the comparison of quantification approaches, were useful in the development of our model for RTD quantification. By utilizing TDQM we could easily identify the counterparts for RTD quantification among the concepts and the comparison matrix increased the efficiency in the process of mapping concepts from the literature to abstracted concepts (Replication Package[3]). However, 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 (see Table VIII).

Relationships between the RTD quantification concepts were also inspired by TDQM as well as derived from the literature found in our SMS. For example, the three RTD types described by Lenarduzzi et al. [19], 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 [25], for example, the relationships between the costs and benefits of rectifying a RTD Item and the Interest incurred by a RTD Item.

The process of extracting concepts from literature, categorizing and grouping them together, and mapping them to abstract concepts was performed by the first author. Concepts and the mappings were examined by the other authors, and where there was disagreement, they were discussed and resolved during meetings. This was done in multiple iterations before the model concepts were finalized. Figure 2 illustrates the concept map for the *RTD Quantification Model (RTDQM)*.

RTD Quantification Concept	Informed by Literature		Informed by TDQM
	density (d)	groundness (g)	TDQM counterpart
User Need	4	4	-
Requirements Engineering Step	3	3	Development Step
Total Cost of a RE Step	3	3	Total Cost of a Dev. Step
(Formalized) Requirement	7	14	Feature
RTD Item	5	12	TD Item
RTD Rectifying Step	1	3	TD Refactoring Step
Cost of Rectifying (or remediating)	4	7	Cost of Refactoring
RTD Interest	5	11	TD Interest
New Code Cost associated with RTD	3	6	New Code Cost associated with TD
Rework Cost associated with RTD	3	6	Rework Cost associated with TD
RE Cost associated with RTD	1	1	-
Benefit of Rectifying	0	0	Benefit of Refactoring
Benefit of taking RTD	1	1	Benefit of taking TD
RTD Interest Probability	1	1	Interest Probability

TABLE VIII
RTD QUANTIFICATION CONCEPTS — D - NUM. OF SOURCES, G - NUM. OF CONCEPTS EXTRACTED FROM A SOURCE THAT RELATES TO A RTD CONCEPT

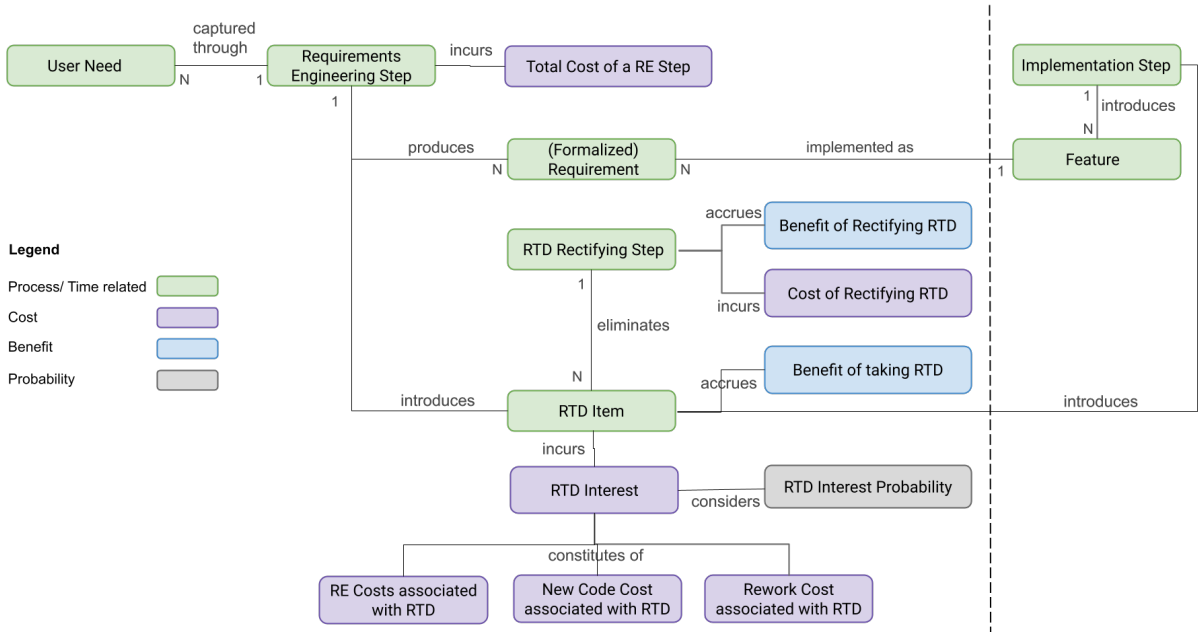

Fig. 2. Left of dashed line: **Concept Map for RTDQM (Our work)**, Right: Connection to TDQM, Perera et al.'s work [25]

RTDQM can be connected to TDQM (See Figure 2). This will complement Perera et al.'s work by showing the modeling of RTD and CTD Quantification together in one model.

Table VIII lists the 14 RTDQM concepts along with two metrics we adopted from Junior et al. [16] to provide some hint on which concepts are grounded in the literature. 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 extracted from a source are related to one RTD quantification concept in our model.

VI. MODELLING RTD QUANTIFICATION — RESULTS (RQ2)

A. Modelling RTD Quantification

The Technical Debt Quantification Model (TDQM) developed by Perera et al. [25], captured the important concepts related to TD Quantification for CTD and illustrated the relationships between those concepts. Modeling RTD quantification in our work resulted in the *RTD Quantification Model (RTDQM)* that captures the important concepts related to RTD quantification and illustrates the relationships between them (See Figure 2 Concept Map for RTDQM, and Table VIII, First column: RTD Quantification Concepts, Last column: code-related counterparts from TDQM [25]).

1) Modelling RE Step and how it connects to the Implementation of Features: User needs for a software product that must be developed are (usually) captured through a Requirements Engineering (RE) Step which incurs a cost, the Total Cost of a RE Step (similar to a Development Step incurring a Total Cost of Development in TDQM). The RE Step produces one to many formalized Requirements. 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. [6]. This is indicated by the relationship between the formalized Requirement and Feature in Figure 2.

2) Modelling RTD Items: Following the 16162 model [4], Perera et al., 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., Requirement Smell, Insufficient or Incomplete requirements, Outdated requirements) [11], [19], [20]. 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. [19] and by Charalampidou et al, as Requirements Documentation Debt [11]) *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 et al., Belle et al., and Abad et al. [1], [6], [14], [19], [20]).

3) Modelling 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, an RTD Item can be eliminated or rectified via a RTD Rectifying 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. [19] describe the Cost of Rectifying RTD as: Cost to formalize and implement the neglected needs, 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. [1], describe it as the Cost of exercising the option.

4) Modelling 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 (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) [1], [7], [14], [19]. 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, for example, RTD Items pertaining to an unused feature will not incur Interest [1].

5) Modelling RTD Interest components: In TDQM, Perera et al., decomposed the Implementation Cost of a Feature and CTD Interest into constituents, New Code Cost and Rework 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 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 Cost associated with RTD model the cascading impact that can incur 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., [19] 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.

However, RTD Interest can have other constituents, such as extra costs incurred during the RE Step that we refer to as 'RE Costs associated with RTD', e.g., the extra cost to perform additional interviews with Users in case the captured User Needs are incomplete. Since Requirements are separate from code (i.e., RE is separate from Implementation), this cost is not represented by New code cost or Rework cost. 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. [20] refer to the cost related to the harmfulness of Requirement Smells, this can possibly incur all three constituents of RTD Interest.

6) Modelling Benefit of Rectifying RTD and Benefit of 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) [15], [17]. However, accumulating RTD can be detrimental in the long run as it could impact the downstream activities such as the development phase as well and in such instances, the developers might end up developing the wrong product for their customer that might have to be scrapped completely [14]. Therefore, it would be more beneficial in the long run to rectify RTD early on, during the early stages of the product development lifecycle prior to implementation. The benefit accrued by rectifying RTD is modeled as 'Benefit of Rectifying RTD', this was informed by the 'Benefit of Refactoring' in TDQM [25] (See Table VIII). However, this concept is not captured in the literature.

VII. Discussion

A. Quantifying CTD vs RTD

The development of RTDQM enabled comparison between the quantification of CTD and RTD (Table VIII, First vs Last

columns). We discuss the observations below.

CTD captures the consequences of sub-optimal decisions made during the development of a software product that concerns the software code [13], [25]. In contrast, RTD captures the consequences of sub-optimal decisions made concerning Requirements. However, RTD can occur during RE or during Implementation (RTD Types 0, 1, 2 [19]). Concepts for both RTD and CTD quantification can be categorized into: process or time, costs, benefits, and probability (themes derived from Perera et al. [25]). These concepts and their relationships help understand TD quantification for both CTD and RTD and, therefore, can be helpful in TDM decision-making for both types of TD. The main differences between the concepts for CTD and RTD can be observed in Table VIII; 'User Need' and 'RE Cost associated with RTD' are the concepts that were identified for RTD in addition to what was informed by TDQM developed for CTD by Perera et al. [25].

Examples of CTD Items include software code-related artifacts such as Code Smells, Design Smells, and Architectural Smells. RTD Items can take the form of ambiguous or low-quality requirements or Requirement Smells (P2, P6, P7), insufficient or incomplete requirements (P3), outdated requirements (P3), neglected or missed user needs (P2), inconsistencies in the SRS (P4), 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) [1], [6], [7], [12], [14], [19], [20]. Like CTD, RTD could also be incurred inadvertently or through deliberate choice (P1, P3) either during RE activities (e.g., when capturing user needs or when specifying requirements in the SRS) or during Implementation (e.g., mismatch implementation, missing features, partially satisfied requirements) [1], [6], [12], [14], [19], [20]. According to Abad et al. (P1) [1], trade-offs in requirements specification can be consequences of intentional, strategic decisions made in pursuit of immediate gains. Charalampidou et al. [11] 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.

The cost to eliminate a CTD Item is referred to as the 'Refactoring Cost' while RTD can have a similar counterpart (P2, P7), the cost to rectify RTD that can be incurred either before or during the implementation stage [14], [19]. If an ambiguous Requirement is corrected during the formalization of Requirements by involving the user to acquire feedback during Requirements validation, the cost is incurred before the Implementation phase. However, if ambiguities in Requirements are not resolved before developing software, rectifying such RTD can be difficult as it might involve challenges such as the difficulties in tracing requirements during the Implementation phase (P3) [11].

Similar to the benefit accrued by taking CTD there can be strategic benefits of taking RTD as well (P1) [1], [2]. However, these benefits can differ, although the end goal (achieving faster delivery to market) is the same for both types of TD. There are also benefits of rectifying RTD, similar to benefits gained by refactoring software code. For example, fixing ambiguous requirements early on can pay off immediately by preventing the impact on development, this benefit is captured by 'Benefit of Rectifying RTD'. However, this concept is not captured in the literature, our model captures it.

CTD Interest impacts the implementation and maintenance activities and does not impact upstream activities such as RE. In contrast, RTD Interest can impact both upstream (e.g., RE activities) and downstream activities (e.g., implementation or maintenance activities) of the software product development life cycle (P1, P2, P3, P4, P7) [1], [11], [14], [19], [20]. RTD can have additional consequences, 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) [19]. RTD Interest incurred during the development stage (P1, P2, P7), e.g., to implement a workaround to compensate for a neglected user need, can be more expensive than resolving such issues earlier in the software development life cycle, which is the downside to accumulating RTD [1], [14], [19]. Furthermore, RTD can 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 [11]. Poorly conducted requirements elicitation (P4, P7) can also lead to building the wrong product that does not meet customer satisfaction [7], [14]. Bonfim et al. (P4) [7] describe this situation as unmet functional and non-functional requirements due to bad specification, and not everything requested is being delivered.

CTD and RTD both involve a probability of incurring Interest (P1), meaning, CTD Items or RTD Items may or may not incur Interest depending on the situation [1]. Some CTD Items might not interfere with the current development and not incur Interest. Similarly, some RTD Items might not impact the current release of the product.

A Development Step in TDQM involves a total cost, the Total Cost of a Development Step [25]. Similarly, a RE Step in RTDQM will incur a Total Cost of RE Step (P1, P2, P4) [1], [7], [19]. However, compared to artifacts related to software code, Requirement artifacts have a feedback loop involving the User to precisely capture User Needs (P2, P3, P6, P7) [11], [14], [19], [20]. Hence, the User is key in determining the optimal set of requirements for developing a software product. This is not the case with the quality of the software code. Delivering value to the customer is the main goal of developing a software product; therefore, managing RTD is equally as important as managing CTD.

However, RTD involves some additional challenges compared to CTD that are inherent to Requirements; these include the inherent uncertainty in Requirements, i.e., requirement changes (P1) and difficulties with traceability of Requirements during the implementation phase (P3) [1], [11]. Lack of trace-

ability increases the effort of keeping requirements updated since it is difficult to identify which requirements can be affected when the source code is changed, and vise-versa [11].

B. Threats to Validity of the Mapping Study

Threats to *identification of primary studies* could apply to the SMS search phase. However, we are confident that we mitigated this by conducting an extensive search in 4 major databases: SCOPUS, ACM, IEEE, and ScienceDirect. The search query was initially developed and tested in SCOPUS before being tailored to other databases. We used keywords, their synonyms and wildcards (*) to capture possible variations of the keywords, plurals and verb conjugations. We applied the search query to the title, abstract, keywords, and full-text to increase the probability of finding all relevant articles and did not limit our search to a particular period. Furthermore, we conducted Forward and Backward Reference Snowballing on the final set of articles so that any missing articles could still be found during snowballing.

C. Threats to Validity of Modelling the Quantification of RTD

RTDQM was informed by the literature found via our SMS conducted focusing on the quantification of RTD and additionally informed by TDQM, Perera et al.'s work [25]. Therefore, we are confident that RTDQM sufficiently captures what is required to model the quantification of RTD. Perera et al.'s work was also informed by literature derived from a SMS and past models of TD (e.g., 16162 model [4]). According to the authors, most of the important parts of their model (concepts and relationships) were captured by many quantification approaches that the model was applied to, in a study they theoretically validated TDQM by applying it to quantification approaches that were not used in the development of the model (e.g., Cost of Refactoring, TD Interest, and TD Item were captured by 21, 21, and 16 TD quantification approaches, respectively) [25]. Their model did not change after that study. Hence, the authors were confident that TDQM sufficiently captured what is required to model the quantification of code-related TD types. In our case, additionally, a case study conducted by Frattini et al. [15] complements our findings. However, RTD is an under-studied research area [19]. Therefore, we see the need to validate our work through case studies which we will do in future work.

Three researchers were involved in the development of RTDQM. The first author conducted the SMS and then developed RTDQM based on the results obtained from the SMS and being informed by TDQM. RTDQM was developed in iterations and was discussed with the other two researchers who validated the SMS results and the concepts and relationships in the model during those iterations. Disagreements were resolved during meetings between the researchers. Another researcher who was not involved in the development of the model did a final review providing feedback. These steps reduced *researcher bias* in the process.

D. Implications to Researchers, Practitioners, Future Work

The main contribution of our work is the RTDQM model that conceptualizes RTD quantification illustrating the concepts related to RTD quantification and the relationships between them. The value of having RTDQM is that it reveals aspects of RTD that are not directly visible. One such aspect is the constituents of RTD Interest; RE Costs associated with RTD, New code cost associated with RTD, and Rework cost associated with RTD. Another aspect is the benefit of rectifying RTD, which is similar (conceptually) to the benefit of refactoring CTD. Another aspect is the feedback loop involved with the user when capturing user needs, different from TD types related to software code. Another use of RTDQM is that it provides a common model to compare different RTD quantification approaches proposed in the literature. The model can also be used as a reference point to develop new quantification approaches.

In future work, we will investigate how practitioners quantify RTD in practice: if they are aware of the costs and benefits associated with RTD (i.e., model concepts) and if metrics support the quantification and, in turn, the management of RTD. We will investigate how practitioners engage in RE, the implementation of features, and if they are aware of the introduction of RTD during these activities and the consequence of accumulating RTD (i.e., RTD Interest). Also, the costs of rectifying, benefits of rectifying, and benefits of taking RTD in situations such as when delivering a MVP. Such a study will increase the awareness of RTD and the importance of RTDM among practitioners while our current work contributes towards the same among researchers.

VIII. Conclusion

We formally defined Requirements Technical Debt (RTD) and developed a conceptual model for RTD quantification. Our work is grounded in the literature obtained via our Systematic Mapping Study (SMS) and informed by prior work. The key observation from the development of our model is that, although RTD is similar in many aspects to TD related to software code (CTD), it has its own components. RTD can incur extra costs (RTD Interest) that impact upstream, e.g., Requirements Engineering (RE), and downstream, e.g., Implementation, Maintenance activities in the software development life cycle. Compared to CTD, RTD has a feedback loop involving the User. Therefore, precisely capturing User Needs and accurately formalizing them as Requirements is important to minimize the accumulation of RTD. It is also essential to monitor and not deviate from the Requirements during Implementation. Furthermore, RTD can incur regardless if there is CTD in the software code. Similar to benefits gained from refactoring software code, rectifying RTD can also accrue benefits. The conceptual model helps compare different RTD quantification approaches and serves as a reference for developing new RTD quantification approaches. In future work, we will investigate if practitioners are aware of the costs and benefits associated with RTD (i.e., model concepts) and how they quantify and manage RTD in practice.

ACKNOWLEDGMENT

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

REFERENCES

[1] Z. S. H. Abad and G. Ruhe, "Using real options to manage technical debt in requirements engineering," in *2015 IEEE 23rd International Requirements Engineering Conference*. IEEE, 2015, pp. 230–235.

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

[3] N. S. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spínola, "Towards an ontology of terms on technical debt," in *2014 Sixth International Workshop on Managing Technical Debt*. IEEE, 2014, pp. 1–7.

[4] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, "Managing technical debt in software engineering (dagstuhl seminar 16162)," in *Dagstuhl Reports*, vol. 6, no. 4, 2016.

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

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

[7] V. D. Bonfim and F. B. V. Benitti, "Requirements debt: causes, consequences, and mitigating practices."

[8] V. Braun and V. Clarke, "Using thematic analysis in psychology," *Qualitative research in psychology*, vol. 3, no. 2, pp. 77–101, 2006.

[9] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of systems and software*, vol. 80, no. 4, pp. 571–583, 2007.

[10] A. Cavacini, "What is the best database for computer science journal articles?" *Scientometrics*, vol. 102, no. 3, pp. 2059–2071, 2015.

[11] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "Assessing code smell interest probability: a case study," in *Proceedings of the XP2017 Scientific Workshops*, 2017, pp. 1–8.

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

[13] W. Cunningham, "The WyCash portfolio management system," *Proceedings of the Conference on Object-Oriented Programming Systems, Languages, and Applications, OOPSLA*, vol. Part F1296, no. October, pp. 29–30, 1992.

[14] N. A. Ernst, "On the role of requirements in understanding and managing technical debt," in *2012 Third International Workshop on Managing Technical Debt*. IEEE, 2012, pp. 61–64.

[15] J. Frattini, D. Fucci, D. Mendez, R. Spínola, V. Mandić, N. Taušan, M. O. Ahmad, and J. Gonzalez-Huerta, "An initial theory to understand and manage requirements engineering debt in practice," *Information and Software Technology*, vol. 159, p. 107201, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584923000551

[16] H. J. Junior and G. H. Travassos, "Consolidating a common perspective on technical debt and its management through a tertiary study," *Information and Software Technology*, p. 106964, 2022.

[17] J. Karlsson and K. Ryan, "A cost-value approach for prioritizing requirements," *IEEE software*, vol. 14, no. 5, pp. 67–74, 1997.

[18] B. Kitchenham, R. Pretorius, D. Budgen, O. P. Brereton, M. Turner, M. Niazi, and S. Linkman, "Systematic literature reviews in software engineering–a tertiary study," *Information and software technology*, vol. 52, no. 8, pp. 792–805, 2010.

[19] V. Lenarduzzi and D. Fucci, "Towards a holistic definition of requirements debt," in *2019 ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*. IEEE, 2019, pp. 1–5.

[20] V. Lenarduzzi, D. Fucci, and D. Mendéz, "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. CEUR-WS, 2020.

[21] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management," *Journal of Systems and Software*, vol. 101, pp. 193–220, 2015.

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

[23] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd workshop on managing technical debt*, 2011, pp. 1–8.

[24] J. Perera, "Modelling the quantification of technical debt," in *Companion Proceedings of the 2022 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity*, 2022, pp. 50–53.

[25] J. Perera, E. Tempero, Y.-C. Tu, and K. Blincoe, "Quantifying technical debt: A systematic mapping study and a conceptual model," *arXiv preprint arXiv:2303.06535*, 2023.

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

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

[28] C. Wohlin, "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*, 2014, pp. 1–10.