The Journal of Systems and Software 101 (2015) 193–220

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

A systematic mapping study on technical debt and its management

Zengyang Li[a,*], Paris Avgeriou[a], Peng Liang[b,c]

[a] Department of Mathematics and Computing Science, University of Groningen, Nijenborgh 9, 9747 AG Groningen, The Netherlands
[b] State Key Lab of Software Engineering, School of Computer, Wuhan University, Luojiashan, 430072 Wuhan, China
[c] Department of Computer Science, VU University Amsterdam, De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

ARTICLE INFO

Article history:
Received 11 July 2014
Revised 10 December 2014
Accepted 10 December 2014
Available online 16 December 2014

Keywords:
Systematic mapping study
Technical debt
Technical debt management

ABSTRACT

Context: Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system.
Objective: This work aims at collecting studies on TD and TD management (TDM), and making a classification and thematic analysis on these studies, to obtain a comprehensive understanding on the TD concept and an overview on the current state of research on TDM.
Method: A systematic mapping study was performed to identify and analyze research on TD and its management, covering publications between 1992 and 2013.
Results: Ninety-four studies were finally selected. TD was classified into 10 types, 8 TDM activities were identified, and 29 tools for TDM were collected.
Conclusions: The term "debt" has been used in different ways by different people, which leads to ambiguous interpretation of the term. Code-related TD and its management have gained the most attention. There is a need for more empirical studies with high-quality evidence on the whole TDM process and on the application of specific TDM approaches in industrial settings. Moreover, dedicated TDM tools are needed for managing various types of TD in the whole TDM process.

1. Introduction

Technical debt (TD) is a metaphor reflecting technical compromises that can yield short-term benefit but may hurt the long-term health of a software system. This metaphor was initially concerned with software implementation (i.e., at code level), but it has been gradually extended to software architecture, detailed design, and even documentation, requirements, and testing (Brown et al., 2010). Although the technical debt metaphor was proposed two decades ago, it has only received significant attention from researchers in the past few years.

TD can do both good and harm to a software project. TD that is intentionally incurred (to achieve some short-term benefit) can be fruitful (Allman, 2012) if the cost of the TD is kept visible and under control. In some cases, the development team may choose to take some TD in order to obtain business value. For instance, incurring TD can speed up the development of new features, thus helping the company move ahead of competition. On the other hand, TD can also be incurred unintentionally, meaning that the project manager and development team are not aware of the existence, location, and consequences of the TD. If left invisible and unresolved, TD can be accumulated incrementally, which in turn results in challenges for maintenance and evolution tasks.

Both intentional and unintentional TD (McConnell, 2008) should be managed in order to keep the accumulated TD under control (Lim et al., 2012). TD management (TDM) includes activities that prevent potential TD (both intentional and unintentional) from being incurred, as well as those activities that deal with the accumulated TD to make it visible and controllable, and to keep a balance between cost and value of the software project.

In order to systematically manage TD, it is necessary to have a clear and thorough understanding on the state of the art of TDM. Different methods and tools have been used, proposed, and developed for TDM, but it is not clear how these methods and tools map to TDM activities. Furthermore, although TD has gained significant attention over the past years, the researchers and practitioners in the TD community perceive the concept of TD in different ways, while ambiguities exist around the inevitable hype of TD. For example it is still unclear what can be classified as TD and what cannot in software development, the compromise of which system quality attributes is considered as TD, and what are the limits of the TD metaphor. Answering these basic questions on the TD concept would help researchers to advance the state of the art and practitioners to appraise and select techniques for TDM in their application context.

In this paper, we report the results of a systematic mapping study broadly examining the concept of TD and its management. A

* Corresponding author. Tel.: +31503633968.
E-mail address: zengyangli@gmail.com (Z. Li).

http://dx.doi.org/10.1016/j.jss.2014.12.027
0164-1212/© 2014 Elsevier Inc. All rights reserved.

systematic mapping study is a form of secondary study aiming to get a comprehensive overview on a certain research topic, to identify research gaps, and to collect evidence in order to direct future research (Kitchenham and Charters, 2007, Engström and Runeson, 2011). It allows all available studies in a domain to be analyzed at a high level thereby answering broad research questions regarding the current state of the research on a topic (Kitchenham and Charters, 2007). Another form of secondary study is a systematic literature review (SLR), which aims at identifying, evaluating, and interpreting all available studies to answer particular research questions, and requires more in-depth analysis (Kitchenham and Charters, 2007). We selected to conduct a mapping study instead of a SLR because the involved domain of TD is quite broad and we want to include all research literature (excluding gray literature) in the domain and classify it. Our focus is thus not on analyzing particular aspects of the involved domain, but on answering broad questions about the overall domain. This mapping study on TD and its management has the following objectives:

(1) To get a comprehensive understanding of the concept of "technical debt" in software development based on existing research work on TD;
(2) To get an overview of the current state of the research on TDM, including TDM activities, approaches, and tools;
(3) To identify promising directions for future research on TD and its management.

Despite the significant attention to the field of TD, our systematic mapping study indicates that there are no other secondary studies that comprehensively investigate the concept of TD and its management. One other significant work in this area, by Tom et al., reports on a study for understanding the dimensions of TD, the reasons for incurring TD, and the benefits and drawbacks of allowing TD to accrue (Tom et al., 2013). It involves a multivocal literature review (MLR) and is supplemented by interviews with software practitioners and academics to establish the boundaries of the TD phenomenon (Tom et al., 2013). The MLR is based on their previous SLR on TD (Tom et al., 2012), in the sense that the results of that SLR are combined into the MLR. Our mapping study and Tom et al.'s work are complementary to each other, in the following three aspects:

- **Objectives**. Our mapping study mainly aims to get (1) a comprehensive understanding on the concept of TD; (2) an overview of the current state of the research on TDM; (3) promising future research directions. In contrast, the work of Tom et al. (2013) focuses on the dimensions and causes of TD, and the benefits and drawbacks of allowing TD. The first objective of our study has a partial overlap with the study of Tom et al. (2013): the types of TD in our study are similar to the dimensions of TD in Tom et al.'s work. However, we collected more types of TD than the dimensions of TD in Tom et al.'s work, and we further classified the TD types into sub-types of TD. We also investigated several other aspects of TD that were not studied in Tom et al.'s work, including the studied and under-studied TD (sub-)types, TD-related notions, the compromised quality attributes when TD is incurred, and the limits of the TD metaphor. In contrast, Tom et al. looked into the reasons for TD, and the benefits and drawbacks of incurring TD, which were not investigated in our work. Thus, our mapping study and the work of Tom et al. are complementary to each other for the purpose of covering the whole research field of TD.
- **Methodology**. Both are secondary studies on the topic of TD. Our work applied a systematic mapping study method, while the work of Tom et al. used a SLR and MLR. The differences between the systematic mapping study and SLR methods are the type of research questions asked and analysis conducted on the literature review (Kitchenham and Charters, 2007). As aforementioned, a systematic mapping study provides demographics and classifications to answer broad research questions about a particular topic, while a SLR provides in-depth analysis to answer more specific research questions of the topic investigated (Kitchenham and Charters, 2007).
- **Primary studies**. First, our mapping study examined the research work published from 1992 to 2013, while the work of Tom et al. (2013) systematically checked the publications before 2011 (they conducted the SLR in 2011). Second, our mapping study only includes peer-reviewed publications as primary studies, while Tom et al. (2013) includes both peer-reviewed publications and web blogs and articles. Third, our mapping study selected 94 peer-reviewed primary studies, compared with 19 peer-reviewed primary studies in Tom et al. (2013); this can be partly explained by the large number of studies on TD published in the last two years as shown in Fig. 5. In addition, the study of Tom et al. also included around 35 web blogs and articles. In the work of Tom et al., they described that they also reviewed the papers published in the Managing Technical Debt (MTD) workshops in 2010–2012, and in the IEEE Software special issue on TD (November/December 2012), but they included these papers as additional sources instead of primary studies. Thus, the set of the primary studies in our mapping study is significantly different from that of Tom et al.'s.

The remainder of this paper is structured as follows: Section 2 describes the research questions of this systematic mapping study. The procedure of this mapping study is detailed in Section 3. Section 4 presents the synthesis results of the extracted data from the selected studies and answers the research questions. Section 5 discusses the mapping study results and their implications to researchers and practitioners. Section 6 discusses the threats to validity of this mapping study, and Section 7 presents the conclusions drawn in this mapping study.

2. Research questions

The goal of this study, described using the Goal-Question-Metric approach (Basili, 1992), is: to analyze *primary studies on TD* for the purpose of *getting a comprehensive understanding* with respect to the *TD concept and TDM*, from the point of view of *researchers and practitioners* in the context of *software development*.

To achieve the objectives presented in Section 1, this mapping study will answer the following research questions (RQs) classified into two categories below. The answers to these two categories of RQs can be linked to the objectives of this mapping study: a comprehensive understanding of the concept of TD (Category 1 of RQs), an overview of the current research on TDM (Category 2 of RQs), and promising future research directions on TD and TDM (Categories 1 and 2 of RQs).

(1) RQs on the TD concept
The RQs in this category concern the overall concept of TD. The answers to these RQs can provide us with a comprehensive understanding on TD.
RQ1: *What are the types of TD and what is not considered as TD?*
Rationale: A TD type refers to a specific category of TD (e.g., architectural, design, code) or a sub-category based on the cause of TD (e.g., architectural TD can be caused by architecture smells). TD can also be classified in other ways, such as strategic and non-strategic TD, or the TD quadrant (Fowler, 2009). However, in this mapping study we focus on the classification of TD according to the phases of the software development lifecycle, as this can help stakeholders in different roles (e.g., requirements engineer, architect, test engineer) become aware of what TD may be incurred during the development phases that they are involved. By answering this RQ, we can list the types of TD and potentially shed light on some conflicting viewpoints on these types. In addition, not all things that are detrimental

to a software system are TD. We use the term non-TD to denote such aspects that are detrimental to a software system but are not considered as TD. We can also get a clearer boundary between TD and non-TD.

RQ2: *What TD types are researchers and practitioners mostly working on and what types are under-studied?*

Rationale: Not all types of TD have received equal attention from the TD community. For instance, the TD types that can be detected by existing tools may have received more attention. By answering this RQ, we can get information about which types of TD (types are derived from answering RQ1) are most interesting for researchers and practitioners, and what TD types are left under-studied. The latter may point out research gaps and thus lead to future research directions.

RQ3: *What notions are used for describing and explaining TD?*

Rationale: A notion here refers to any term that has direct relationship with TD and is used to describe or explain the concept of TD. Such a notion should make sense for TD in general, and not only for a specific type of TD (e.g., requirements TD). Examples of such notions are *principal* and *interest*. The extracted notions do not overlap with the extracted TD types in RQ1. They are two different concerns in this mapping study. The former emphasizes the categorization of TD instances while the latter focuses on the explanation and description of the TD phenomenon as a whole. By answering this RQ, we can obtain the vocabulary directly related to TD and notions that tend to be accepted in the community of TD.

RQ4: *Which quality attributes are compromised when TD is incurred?*

Rationale: TD makes compromises on system quality attributes, such as maintainability. The answer to this RQ will provide us with a list of quality attributes that are compromised when TD is incurred. This list of quality attributes and the frequencies that those quality attributes are mentioned in the selected studies, can to some extent reflect the understanding of researchers and practitioners on the scope of TD, i.e., which quality attributes are considered relevant to TD and which ones not.

RQ5: *What are the limits of the TD metaphor?*

Rationale: The TD metaphor may not be a perfect metaphor. We want to know what limits the metaphor has, e.g., Rooney argues that the TD metaphor is not perfectly applicable for development approaches like Scrum) in which code is continuously improved and therefore TD is continuously repaid (Rooney, 2010). This information can help us deepen the understanding of this metaphor, and inspire us to refine further the TD metaphor and its management extending the original concept from the financial domain.

(2) RQs on TDM

The RQs in this category focus on the research on TDM. The answers to these RQs will provide an overview on TDM activities, approaches, and tools, as well as challenges in TDM research and application.

RQ6: *What are the different activities of TDM?*

Rationale: By answering this RQ, we can understand what activities are performed in TDM and what activities are mostly discussed or used to manage TD (e.g., TD identification and measurement) in software development.

RQ7: *What approaches are used in each TDM activity?*

Rationale: By answering this RQ, we can get an overview of the approaches that have been proposed, developed, and employed for different TDM activities. This answer can inform practitioners what approaches they can use in specific TDM activities, and also help researchers to identify the research gaps in approaches for various TDM activities.

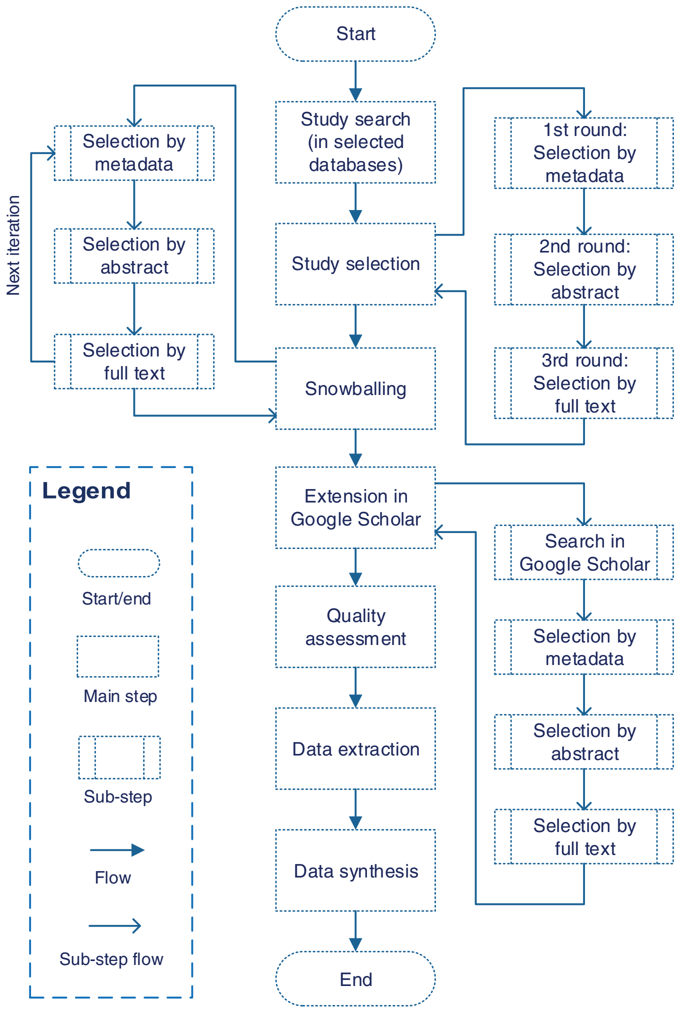

Fig. 1. The procedure of this mapping study.

RQ8: *What tools are used in TDM and what TDM activities are supported by these tools?*

Rationale: Appropriate tools can facilitate the management of TD. The answer of this RQ can help practitioners in selecting available tools for different TDM activities, and also help researchers to adapt or develop new tools for TDM.

RQ9: *What challenges for TDM have been identified?*

Rationale: The answer to this RQ will provide a list of challenges identified by the TD researchers. Subsequently this can lead us to focus future research and to identify issues in the application of TDM techniques in practice.

3. Mapping study execution

The procedure of this mapping study follows the guidelines for performing secondary studies (including SLR and systematic mapping study) proposed by Kitchenham and Charters (2007). Although there are dedicated guidelines for performing mapping studies (i.e., Petersen et al., 2008), we did not use them in this study since some RQs cannot be answered only by mappings (see details about data synthesis in Section 3.7). Similar to Li et al. (2013), we do not use study quality assessment results to filter primary studies. The execution procedure of this mapping study is shown in Fig. 1. The details of the steps in the procedure are presented in the following subsections.

Table 1
Electronic databases searched.

#	Database	Selected
DB1	IEEEXplore	Yes
DB2	ACM Digital Library	Yes
DB3	Science Direct	Yes
DB4	ISI Web of Science	Yes
DB5	SpringerLink	Yes
DB6	Scopus	Yes
DB7	Inspec	Yes
DB8	CiteSeer	No (after trial search)
DB9	Wiley InterScience	No (after trial search)

Table 2
Use of selection criteria in study selection.

Selection round	Criteria used
1st Round: selection by metadata	I1, I3
2nd Round: selection by abstract	I1, I2, E1
3rd Round: selection by full text	I1, I2, E1

3.1. Study search

We used an automatic search method to retrieve the relevant studies for this mapping study in the searching phase. We retrieved studies in a number of selected electronic databases, with the search terms described in Section 3.1.2, through the search engines provided by the databases. Manual search method on target venues (e.g., journals and conferences) was not used in this mapping study, since there are no specific venues for this study topic, except for the MTD workshops. But all the papers published in the MTD workshops can be retrieved in IEEEXplore database, which is one of the databases included in the automatic search. We present the details of the automatic search in the following subsections.

3.1.1. Search scope

The search scope is important to a mapping study since it influences the needed effort for the study search and the completeness of primary studies that are potentially relevant to the research topic. The search scope of this mapping study includes the time period and electronic databases.

3.1.1.1. Time period. We chose the year 1992 as the start of the search period for this mapping study since the TD metaphor was coined that year (Cunningham, 1992). The end of the search period is December 2013 when we started this mapping study.

3.1.1.2. Electronic databases. Study searches were performed in nine main electronic databases, as suggested in Chen et al. (2010), which are listed in Table 1. Although EI Compendex is considered as an important source (Kitchenham and Charters, 2007, Chen et al., 2010), we did not include this database for the following reasons: (1) EI Compendex is inaccessible in our universities; (2) most publication venues indexed in EI Compendex are also indexed in other selected electronic databases; and (3) the "snowballing" and extension in Google Scholar, which are introduced later in Sections 3.3 and 3.4, largely include the missed relevant studies which were only indexed in EI Compendex (if any).

3.1.2. Search strategy

The search strategy of the mapping study affects the completeness of the retrieved studies and the effort that we need to spend, thus, it should be carefully designed. The search strategy is detailed as follows:

(1) Trial searches were performed in each database in Table 1 with the intention of checking the number of returned papers from each database. The objective of the trial searches is to check whether it makes sense to spend effort in all the databases. We used "technical debt" as the search string in the abstract to search studies in the electronic databases. Multiple peer-reviewed publications on TD were found in DB1–DB7, while only one paper (a technical report, which is not peer-reviewed and should be excluded, see inclusion criterion I3 in Section 3.2.1) was returned in DB8 and no TD related study was found in the returned results of DB9. Therefore, we did not include DB8 and DB9 as study sources for automatic searches, as noted in Table 1.
(2) The search string was adjusted to the single term "debt" for formal automatic searches, because some papers related to TD do not use the term "technical debt" explicitly but use the name of a specific form of technical debt, such as, design debt, code smells debt. This helps to maximize the number of the returned relevant papers, as it places as few restrictions as possible on the search string. Note that, the automatic searches using database search engines should be limited in the area of computer science or software engineering, depending on whether domain search option is provided by the search engines of the databases.
(3) Formal automatic searches with the search string "debt" were performed in DB1–DB7.

3.2. Study selection

To ensure that the study selection results are objective, we defined selection criteria (Section 3.2.1) that were employed in the study selection process (Section 3.2.2).

3.2.1. Selection criteria

The following criteria were used as inclusion criteria:

I1: The paper is related to software development. Papers on financial debt should not be included.
I2: The paper should focus on some specific types of TD or TD as a whole. If a paper just mentions the concept of TD, without in-depth investigation or detailed explanation in the context of that paper on specific types of TD, TD occurring in actual cases, or the concept of TD, then this paper should not be included.
I3: The paper should be peer-reviewed, i.e., published in journals, conference proceedings, workshop proceedings, or book chapters. A publication that has not undergone a peer review is considered informal and not included.

Exclusion criterion, for this mapping study is:

E1: Any paper published in the form of abstract, tutorial, or talk is excluded. Papers in the form of abstract, tutorial, or talk do not provide enough details that are required in scientific papers to illustrate the research question and its solution in depth.

The language of papers is not regarded as an exclusion criterion, since we had filtered out the non-English papers by adjusting the settings of the search engines of the selected databases during the study search step.

3.2.2. Selection process

The study selection includes the following steps:

(1) First round study selection. One researcher filtered papers based on **metadata** including title, keywords, and venue name, applying the inclusion criteria I1 and I3 as shown in Table 2. The inclusion criteria I2 and E1 were not used in this round, since it is impossible to know whether a paper focuses on certain type of TD and the form of a paper (e.g., tutorial) according to the

metadata only. A paper with any doubt about its relevance to the study topic was included for the next round selection.

(2) Second round study selection. Two researchers independently filtered papers by reading the **abstracts** of the papers left in the first round selection and applying the criteria I1, I2, and E1. The criterion I3 was inapplicable in this round since we can identify whether a paper is peer-reviewed in the first round and we do not need to do it again. Selection results were verified by two researchers and any disagreements on the selection results were discussed and addressed. If the disagreement could not be resolved, the paper was included. If it was difficult to make a decision on whether a paper should be included or not, the paper was included.

(3) Third round study selection. Two researchers independently filtered papers by reading the **full text** of the papers left in the second round selection, applying the criteria I1 and E1 as shown in Table 2. For the same reason stated in the previous bullet, the criterion I3 was inapplicable in this round. Again, any disagreements on selection results were discussed and consensus on the final selection results was achieved.

3.3. Snowballing

In order not to miss any potentially relevant studies, we applied the "snowballing" technique to find more potentially relevant studies by checking the references of each selected study (Budgen et al., 2008, Wohlin, 2014). Snowballing is an iterative process: the first iteration uses as input the selected studies from the study selection phase and checks their referenced papers; in subsequent iterations, the referenced papers of the newly selected studies of the last iteration are checked; this iterative process ends at an iteration during which no study is newly selected. Each iteration follows the three study selection sub-steps, exactly like the study selection phase as described in Section 3.2.2: based on metadata, on abstracts, and on full text. The selected studies from the snowballing process were combined into the final results of the study selection.

3.4. Extension in Google Scholar

To ensure the set of selected studies are complete and we do not miss relevant studies, we extended the study search in Google Scholar (GS). Specifically, we checked against the top 200 results of the search with the search string "technical debt" in Google Scholar. The reasons we used the search string "technical debt" instead of "debt" as in the formal automatic searches are that (1) too many publications on other types of debt (e.g., financial debt) were returned and (2) there was no effective way to restrict publications to the software engineering field when using "debt" as the search string in Google Scholar. First we selected the potentially relevant papers that are not in the set of the selected studies through the previous steps (study selection and snowballing), and then the papers also underwent filtering based on metadata, abstract and full text (just like we did for study selection and snowballing).

We did not choose Google Scholar as one of the databases for study search for the following reasons: (1) most of the returned papers in Google Scholar are indexed in the selected databases; (2) we cannot limit the search of "debt" within the software engineering field in Google Scholar, which causes the number of returned results to be largely beyond what can be manually checked; and (3) the returned papers in Google Scholar are sorted by Google's PageRank technique, which may cause recently published relevant papers to rank very low.

3.5. Quality assessment

All the finally selected studies underwent quality assessment through a set of questions regarding the evidence level of the study and the quality of the data items to be extracted. The quality assessment questions are described as follows:

Q1: How much evidence supports the claims related to TD in the study? We adopted the evidence hierarchy proposed in (Alves et al., 2010). More specifically, the evidence hierarchy is defined as follows (from weakest to strongest):
  Level 0: No evidence.
  Level 1: Evidence obtained from demonstration or working out toy examples.
  Level 2: Evidence obtained from expert opinions or observations.
  Level 3: Evidence obtained from academic studies, e.g., controlled lab experiments.
  Level 4: Evidence obtained from industrial studies, e.g., causal case studies.
  Level 5: Evidence obtained from industrial practice.
Q2: Is there a clear statement of the aims of the research?
Q3: Is there a clear statement of the definition of TD (defined by the authors or adopted from other references)?
Q4: Is there a clear statement of which types of TD the paper focuses on?
Q5: Are the limitations of the study discussed explicitly?

Question Q1 evaluates the evidence level of the study related to the TD concept and its management approaches in the selected paper. In particular, we make a distinction between "evidence obtained from expert opinions" and "evidence obtained from industrial practice". If a paper only presents opinions of industrial experts or academic researchers without supported empirical studies, we consider that the paper provides evidence from expert opinions. The evidence from industrial practice indicates that the claims related to TD has already been approved and adopted by some industrial organizations for daily engineering practice (Alves et al., 2010, Galster et al., 2014). Q2 and Q5 were adopted from Dybå and Dingsøyr (2008) and Ali et al. (2010) while Q3 and Q4 are formulated according to our study topic and RQs. We adopted and adjusted the grading rules for the quality assessment questions used by Dybå and Dingsøyr (2008). A three-point scale is used to answer questions Q2, Q3, Q4, and Q5, i.e., "yes", "to some extend", and "no". Each quality assessment question was further quantified by assigning a numerical value to each answer ("yes" = 1, "to some extend" = 0.5, and "no" = 0). For question Q1, a six-point scale is used to grade the six evidence levels (from weakest to strongest evidence, the score of the evidence level of each selected study can be 0.0, 0.2, 0.4, 0.6, 0.8, or 1.0). Then, a quality assessment score can be given to a study by summing up the scores to all the questions for the study.

3.6. Data extraction

To answer RQs as presented in Section 2, we extracted the data items listed in Table 3 from each selected study. The extracted data were recorded on a spreadsheet.

Before data extraction, we discussed the definitions of the data items to be extracted to clarify the meanings of the data items to all the authors. To make sure that all the authors have the same understanding on the data items, before the formal data extraction, two authors did a pilot data extraction with five studies and all disagreements were discussed and resolved. After the pilot data extraction, one author extracted data from part of the selected studies and another author performed the data extraction on the rest of the selected studies independently. Finally, the two authors checked all the extracted data together to make sure that the data are valid and clear for further analysis.

Table 3
Data items extracted from each study.

#	Data item name	Description	Relevant RQ
D1	Year	The publication year of the study	None
D2	Venue	The name of the publication venue of the study	None
D3	Publication type	Journal, conference, workshop, or book chapter	None
D4	Author type	Industry, academia, or both	None
D5	TD type	The type of TD, e.g., Architectural or code TD	RQ1
D6	Non-TD	The concepts that are not regarded as TD, e.g., unimplemented features	RQ1
D7	TD studied	TD types that researchers and practitioners are mostly working on	RQ2
D8	TD-related notions (notions relevant to TD)	The notions (e.g., interest) that have direct relationships with TD, and are used or discussed in the study. There should be clear definitions or descriptions about the notions, or the notions are self-explanatory. The notions should make sense in general and not only used to describe or explain a specific form of TD or a certain TDM approach. For a selected study, the extracted content of this data item should not have overlaps with the data items TD type (D8) and TDM activity (D12)	RQ3
D9	Quality attributes compromised	The quality attributes that are compromised by incurring TD	RQ4
D10	Limit	The limits of the TD metaphor when applied in the software engineering domain	RQ5
D11	TDM activity	The TDM activities that are discussed in the study, e.g., TD identification and measurement	RQ6
D12	TDM approach, TD types it handles, and its input artifacts	The TDM approach that conducts specific TDM activities, the TD types that the TDM approach deals with, and the artifacts that the TDM approach takes as input. A TDM approach is not limited to be used for one specific TDM activity, but might be used for multiple TDM activities	RQ7
D13	TDM tool, its functionalities, TD types it handles, its input artifacts, and supporting TDM activities	The TDM tool used in the study, the functionalities of the tool, the TD types handled by the tool, the artifacts taken as input by the tool, and the TDM activities supported by this tool	RQ8
D14	Challenges	The challenges on TDM that have been identified in the study	RQ9

3.7. Data synthesis

Data synthesis aims to synthesize the extracted data to answer the RQs defined in Section 2. Descriptive statistics and frequency analysis were employed in synthesizing the data to answer RQ3, RQ6, RQ7, and RQ8. When synthesizing the data to answer RQ7, besides using descriptive statistics, we also plotted the relevant studies to a map which has three dimensions (i.e., TDM activity, publication year, and TD type). This map provides the distribution of all the selected studies in which TDM approaches are mentioned, and we also categorized concrete approaches for each TDM activity. In answering RQ8, in addition to using descriptive statistics, we also tabulated the collected tools, their related information (e.g., related studies and TD types handled), and their supported TDM activities. To answer RQ1 and RQ2, we created a classification tree that categorizes TD into different types, and each type was further categorized into sub-types based on the causes of the TD type. To answer RQ4, we mapped the collected quality attributes that are considered compromised when TD is incurred, to the software product quality model proposed in ISO/IEC 25010 [2011], a uniform model of software quality attributes. As for RQ5, we just list the collected data as we only found several studies that explicitly discuss the limits of TD. In constructing the answer for RQ9, we categorized the challenges into different types according to the topics of the challenges. The data synthesis of this mapping study is further detailed in Sections 4.4 and 4.5, along with the study results to the RQs.

It is worth explaining how we synthesized the TD-related notions (data item D8 in Table 3) because the synthesis of this data item is more subjective than of other data items. As described in data item D8 in Table 3, we distinguish TD-related notions from TD types and TDM activities. A TD-related notion is used to explain or describe the TD concept in general. We followed the rules described below:

- If a candidate notion describes some aspects of TD (e.g., property and uncertainty), then keep it.
- If a candidate notion describes TD in general instead of a specific type of TD (e.g., test TD), then keep it.
- If a candidate notion can only be used to describe a specific type of TD (e.g., test TD), then remove it.
- If a candidate notion is a synonym to another candidate notion, then merge them into one.
- If a candidate notion *A* describes a special case of another candidate notion *B*, then keep notion *B* (the general type) and remove notion *A*. For instance, "compounding interest" is a type of "interest", so we just keep the latter and remove the former.

4. Study results

We performed this systematic mapping study according to the procedure described in Section 3. We first present the study search and selection results in Section 4.1, then describe the demographic data of the selected studies in Section 4.2 and the study quality assessment results in Section 4.3, and finally answer the RQs defined in Section 2: Section 4.4 answers the first category of RQs (RQ1, RQ2, RQ3, RQ4, and RQ5); Section 4.5 answers the second category of RQs (RQ6, RQ7, RQ8, and RQ9).

4.1. Search and selection results

Fig. 2 shows the study search results, the selection results in each round of study selection, as well as the selection results from the references check (snowballing) and the study extension by Google Scholar. Totally, 1665 papers were returned from the database searches (described in section 3.1.1.2), 168 papers were left after the selection by metadata (1st round), 107 papers left after duplicated papers were removed, 94 papers left after the selection by abstract (2nd round), and 75 papers left after the selection by full text (3rd round). 10 more papers were selected after checking the references (snowballing) of the 75 selected papers. Nine more papers were selected after the study extension by Google Scholar, during which we also performed snowballing based on nine selected papers and no extra related papers found (this snowballing is not shown in both Figs. 1 2). Thus, in total, there are 94 primary studies finally selected in this mapping study (see Appendix A).

4.2. Demographic results

This section describes the demographic data of the selected studies, i.e., the study classifications by author type, publication venue, and publication year.

4.2.1. Classification by author type

As shown in Fig. 3, authors of 43% of the selected studies (i.e., 40 studies) work in industry, and authors of 40% of the selected studies

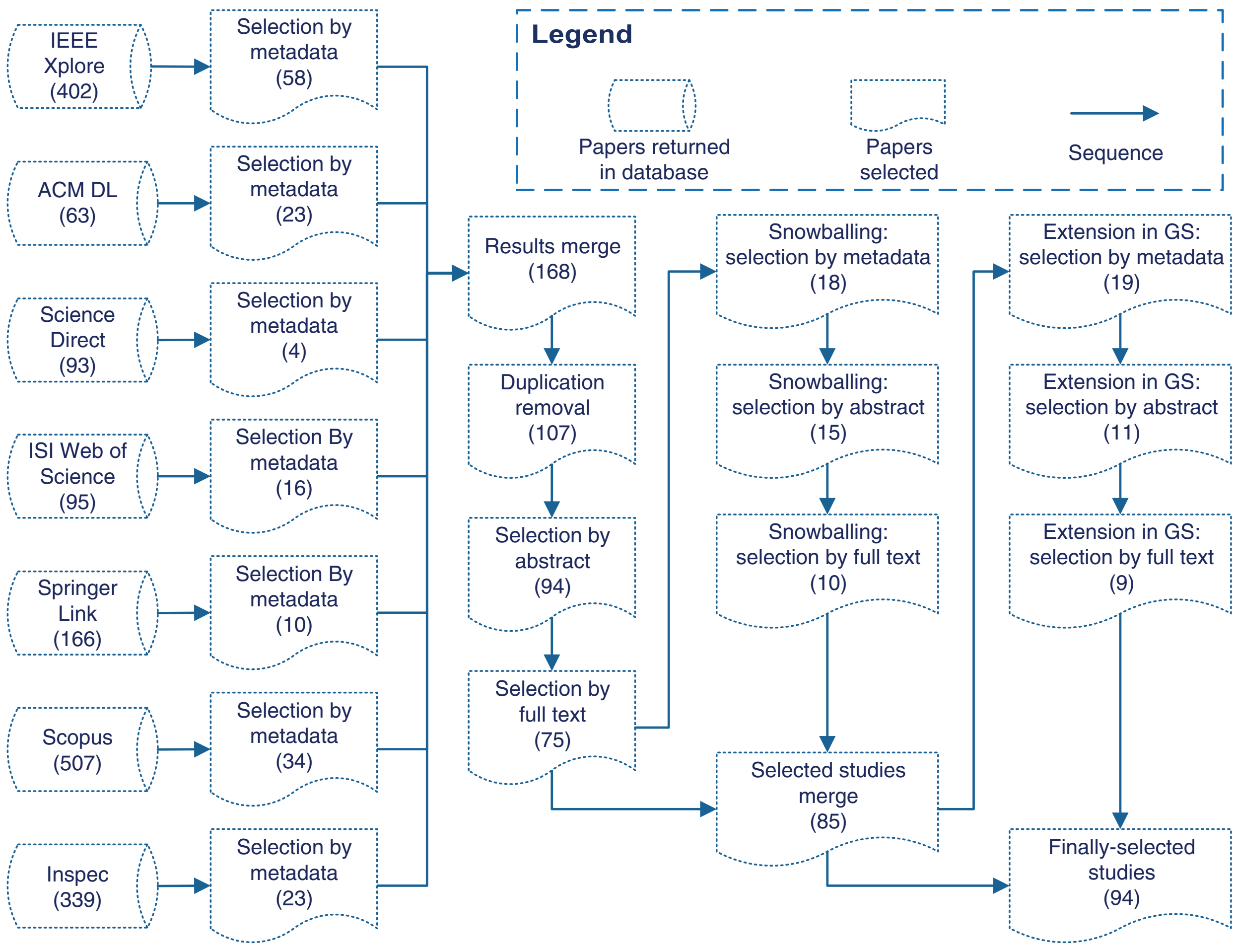
Legend
Papers returned in database
Papers selected
Sequence
IEEE Xplore (402)
Selection by metadata (58)
ACM DL (63)
Selection by metadata (23)
Science Direct (93)
Selection by metadata (4)
ISI Web of Science (95)
Selection By metadata (16)
Springer Link (166)
Selection By metadata (10)
Scopus (507)
Selection by metadata (34)
Inspec (339)
Selection by metadata (23)
Results merge (168)
Duplication removal (107)
Selection by abstract (94)
Selection by full text (75)
Snowballing: selection by metadata (18)
Snowballing: selection by abstract (15)
Snowballing: selection by full text (10)
Selected studies merge (85)
Extension in GS: selection by metadata (19)
Extension in GS: selection by abstract (11)
Extension in GS: selection by full text (9)
Finally-selected studies (94)

Fig. 2. Study search and selection results.

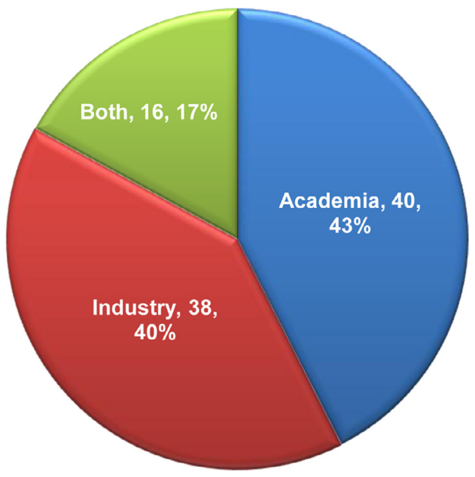

Fig. 3. Distribution of selected studies over author types.

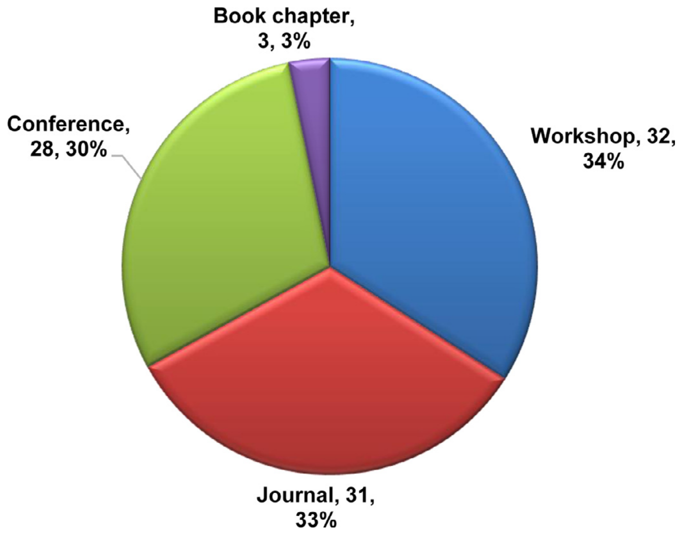

Fig. 4. Distribution of selected studies over publication types.

(i.e., 38 studies) come from academia. The rest (17%) of the selected studies (i.e., 16 studies) come from both academia and industry, i.e., some of the authors of a study work in academia and the other authors work in industry, instead of authors working in both academia and industry simultaneously.

4.2.2. Classifications by publication type and source

Each selected study was published as a conference paper, journal paper, workshop paper, or book chapter. The study distribution over publication types is shown in Fig. 4, in which workshop, journal, and conference are the three, almost equally, popular publication types with 34% (32 studies), 33% (31 studies), and 30% (28 studies) of the selected studies, respectively. Only three studies fall into the publication type of book chapter.

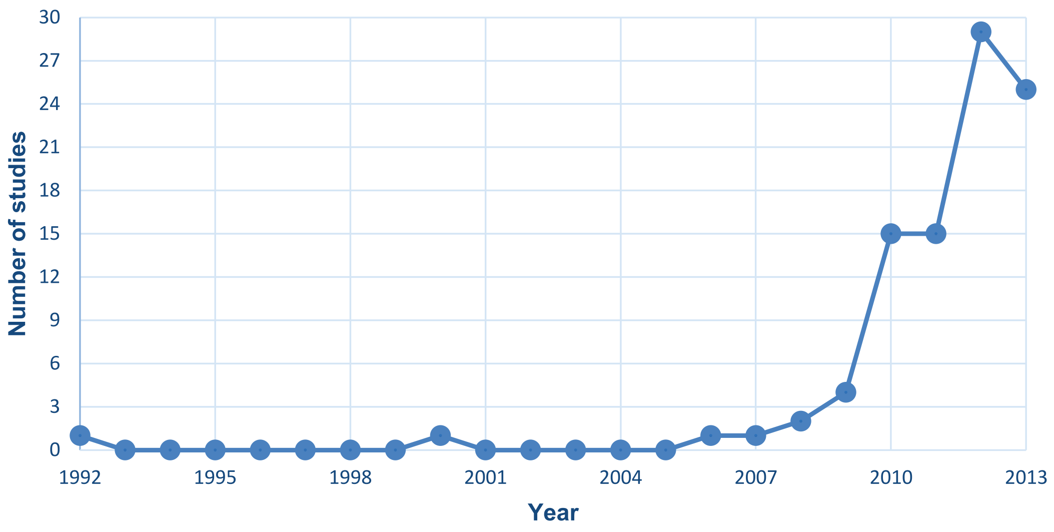

Fig. 5. Distribution of selected studies over time period.

Appendix B presents the publication sources of all the selected studies, their types, number of studies, and the corresponding percentages against the total number of selected studies. The selected studies distribute over 41 publication sources, which indicates that TD and its management have received widespread attention in the entire software engineering community.

4.2.3. Classification by publication year

Fig. 5 shows the distribution of selected studies over the time period from 1992 to 2013. This figure provides clear information on the trend of the number of published studies on TD. During the first 16 years after the TD metaphor was coined, there are no studies published in most of these years, except for only one study per year published in years 1992, 2000, 2006, and 2007. From 2008 to 2013, the number of the published studies on TD has been increasing in general. Especially, since 2010, there were at least 15 studies published per year, which is a big leap compared with the years before 2010. One reason for this could be that the MTD workshop was initiated in 2010 and this workshop raised the attention on TD and the awareness of managing TD.

4.3. Study quality

As stated in Kitchenham and Charters (2007), quality assessment of primary studies is not a necessary task for a mapping study. Thus, we did not use the quality assessment results for study selection, but present them as a reflection of the validity of the selected studies. The quality of the selected studies is evaluated in the quality assessment stage (see Fig. 1) by assigning scores to the quality assessment questions (described in Section 3.5) of each study. The scores of each selected study reflect the quality of the selected study and the credibility of the results of this mapping study (Shahin et al., 2014). The quality assessment results can also indicate potential limitations of current research and provide directions for future research in the field (Kitchenham and Charters, 2007). The detailed quality assessment results are presented in Appendix C. The mean score is 3.17 (out of a maximum of 5.00), which means that the average quality of the selected studies is neither perfect nor unacceptable.

Fig. 6 shows the distribution of the selected studies over quality scores. We can find that most selected studies (61 studies, 65%) received a score in the range (2.50, 4.00]; there are nine selected studies receiving scores falling into the highest score range (4.50, 5.00]; and no selected studies received scores belonging to the lowest score range [0.00, 0.50].

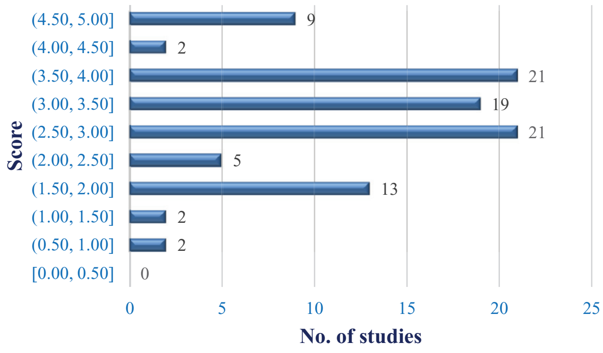

Fig. 6. Distribution of selected studies over quality scores.

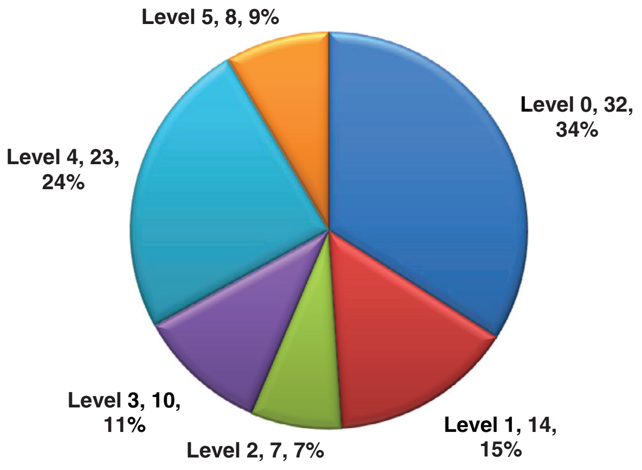

Fig. 7. Distribution of selected studies over evidence levels.

We pay special attention to the evidence level of the claims related to TD in the selected studies (the quality assessment question Q1 in Section 3.5), since it determines with how much confidence, the claims can be trusted. There are six evidence levels (described in Section 3.5) and the higher the evidence level, the more likely a study's claims can be trusted. As presented in Appendix C, the mean score of the evidence level of the selected studies is 0.40 (equal to the score of evidence level 2), which means the average evidence level is relatively low. Fig. 7 shows the distribution of the selected studies over their evidence levels. Thirty-two selected studies (34%) do not have any evidence (level 0), and only eight selected studies (9%) received the highest level of evidence (level 5: evidence obtained from industrial practice).

4.4. TD concept

This section answers the RQs on the concept of TD, i.e., RQ1, RQ2, RQ3, RQ4, and RQ5.

4.4.1. TD and non-TD (RQ1)

RQ1: *What are the types of TD and what is not considered as TD?*

4.4.1.1. TD types. We collected a large number of TD types and instances (each of which can be associated with a TD type) at different levels from the selected studies. These TD types can be classified into 10 coarse-grained types and each of those is further classified into several sub-types based on the causes of TD. The resulting classification tree is shown in Fig. 8.

1. *Requirements TD* refers to the distance between the optimal requirements specification and the actual system implementation, under domain assumptions and constraints (Ernst, 2012).
2. *Architectural TD* is caused by architecture decisions that make compromises in some internal quality aspects, such as maintainability.
3. *Design TD* refers to technical shortcuts that are taken in detailed design.
4. *Code TD* is the poorly written code that violates best coding practices or coding rules. Examples include code duplication and over-complex code.
5. *Test TD* refers to shortcuts taken in testing. An example is lack of tests (e.g., unit tests, integration tests, and acceptance tests).
6. *Build TD* refers to flaws in a software system, in its build system, or in its build process that make the build overly complex and difficult.
7. *Documentation TD* refers to insufficient, incomplete, or outdated documentation in any aspect of software development. Examples include out-of-date architecture documentation and lack of code comments.
8. *Infrastructure TD* refers to a sub-optimal configuration of development-related processes, technologies, supporting tools, etc. Such a sub-optimal configuration negatively affects the team's ability to produce a quality product.
9. *Versioning TD* refers to the problems in source code versioning, such as unnecessary code forks.
10. *Defect TD* refers to defects, bugs, or failures found in software systems.

Note that, for the classification of TD sub-types based on the causes of TD (see Fig. 8), if no causes for a TD type were explicitly specified in a study, we classified the corresponding TD sub-type as "not specified". In Fig. 8, the TD sub-types highlighted with gray are the ones that are studied (not just mentioned, see the difference detailed in Section 4.4.2); a related study marked with an underline is a study that investigates the corresponding TD sub-type in depth. We will elaborate on this further in Section 4.4.2.

As shown in Fig. 8, code TD (as a whole or its sub-types) is the most mentioned TD type in the selected studies (40%, 38 out of 94 studies); architectural TD, test TD, and design TD are the second (27%, 25 studies), third (26%, 24 studies), and fourth (24%, 23 studies) most mentioned TD types respectively; versioning TD, requirements TD, and build TD are the three least mentioned TD types, mentioned in one (1%), three (3%), and three (3%) studies, respectively.

Table 4
Non-TD.

Non-TD	Studies
Defects	S17, S47, S48, S64
Unimplemented features or functionalities	S3, S28, S47, S48
Lack of supporting processes	S85
Unfinished tasks in the development process	S48
Trivial code quality issues	S4
Low external quality	S47

4.4.1.2. Non-TD. A number of selected studies explicitly mention a number of things that should not be regarded as TD. Compared with the number of the TD types that are explicitly mentioned or investigated in the selected studies as shown in Fig. 8, the number of non-TD types is relatively small as listed in Table 4. In particular, only six types of non-TD were collected in the selected studies: defects, unimplemented features, lack of supporting processes, unfinished tasks in the development process, trivial code quality issues, and low external quality. Particularly, in S17, S47, S64, and S48, the authors considered that TD is about the flaws of the internal quality and invisible to external users, which is not the case for defects; thus defects are not TD. In terms of the number of the studies that explicitly mention or investigate what TD is and what it is not, the latter seems to have received little attention from the TD community. The list of non-TD types in Table 4 has not been widely accepted, considering the small number of studies mentioning them. We note that there are four studies claiming that defects are not TD, even though in the previous section, defect TD was identified as TD by 11 studies; we will discuss these two conflicting opinions in Section 5.

4.4.2. TD types studied (RQ2)

RQ2: *What TD types are researchers and practitioners mostly working on and what types are under-studied?*

We mapped the TD types as well as the TD sub-types studied to their related primary studies in Fig. 8. A TD (sub-) type is considered to be studied in a specific primary study if the primary study discussed, or conducted an empirical study on this TD (sub-) type, used an existing TDM approach to manage, and/or proposed a new TDM approach for this (sub-) type of TD. In Fig. 8, the TD sub-types filled with gray are the ones that are studied; a related study marked with an underline is a study that investigates the corresponding TD sub-type in depth.

As shown in Fig. 8, among the 10 types of TD, code TD is the most studied type. Test TD, architectural TD, design TD, documentation TD, and defect TD have also received significant attention. Requirements TD, build TD, infrastructure TD, and versioning TD have not received much attention.

4.4.3. TD-related notions (RQ3)

RQ3: *What notions are used for describing and explaining TD?*

We collected TD-related notions that describe or explain TD. To better understand the relationships between the notions and TD, we used a diagram (see Fig. 9) to show the notions, their categories, their relationships with TD, and the number of studies in which each notion is used. The notions are explained in detail and their related studies are listed in Table 5. We extracted 24 TD-related notions, among which *interest*, *principal*, and *risk* are the top three most used notions. We classify the TD-related notions into five categories according to their meanings and relationships with TD. The notions in the "Metaphor" category are the metaphors borrowed from the economics domain; the notions in the "Property" category describe different properties or characteristics of TD; the notions in the "Uncertainty" category characterize the uncertainty nature of TD; the notions in the "Effect" category are used to describe the effects of TD; the notions in the "Cause" category are related to the causes of how

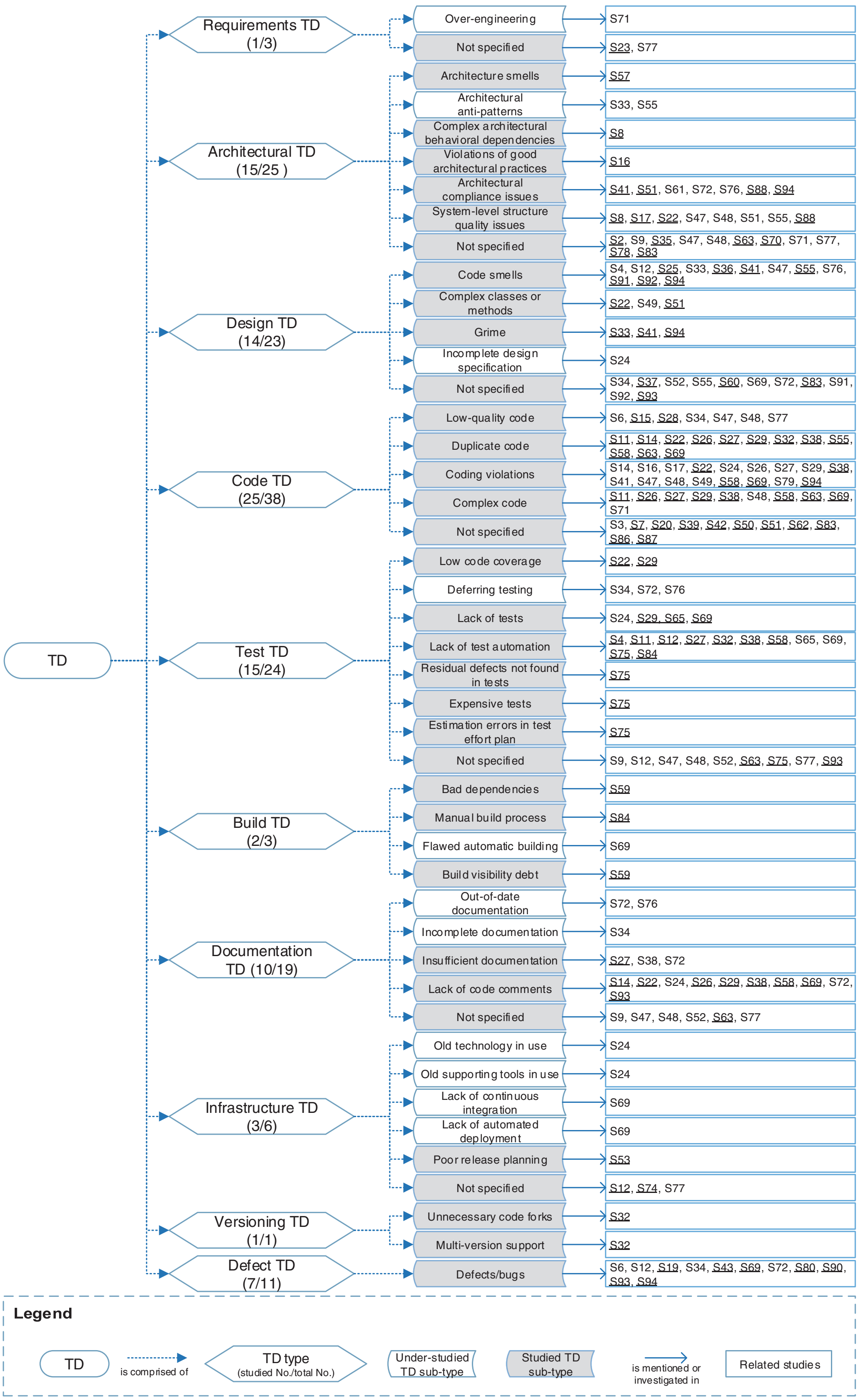
TD
Requirements TD (1/3)
Over-engineering
S71
Not specified
S23, S77
Architectural TD (15/25)
Architecture smells
S57
Architectural anti-patterns
S33, S55
Complex architectural behavioral dependencies
S8
Violations of good architectural practices
S16
Architectural compliance issues
S41, S51, S61, S72, S76, S88, S94
System-level structure quality issues
S8, S17, S22, S47, S48, S51, S55, S88
Not specified
S2, S9, S35, S47, S48, S63, S70, S71, S77, S78, S83
Design TD (14/23)
Code smells
S4, S12, S25, S33, S36, S41, S47, S55, S76, S91, S92, S94
Complex classes or methods
S22, S49, S51
Grime
S33, S41, S94
Incomplete design specification
S24
Not specified
S34, S37, S52, S55, S60, S69, S72, S83, S91, S92, S93
Code TD (25/38)
Low-quality code
S6, S15, S28, S34, S47, S48, S77
Duplicate code
S11, S14, S22, S26, S27, S29, S32, S38, S55, S58, S63, S69
Coding violations
S14, S16, S17, S22, S24, S26, S27, S29, S38, S41, S47, S48, S49, S58, S69, S79, S94
Complex code
S11, S26, S27, S29, S38, S48, S58, S63, S69, S71
Not specified
S3, S7, S20, S39, S42, S50, S51, S62, S83, S86, S87
Test TD (15/24)
Low code coverage
S22, S29
Deferring testing
S34, S72, S76
Lack of tests
S24, S29, S65, S69
Lack of test automation
S4, S11, S12, S27, S32, S38, S58, S65, S69, S75, S84
Residual defects not found in tests
S75
Expensive tests
S75
Estimation errors in test effort plan
S75
Not specified
S9, S12, S47, S48, S52, S63, S75, S77, S93
Build TD (2/3)
Bad dependencies
S59
Manual build process
S84
Flawed automatic building
S69
Build visibility debt
S59
Documentation TD (10/19)
Out-of-date documentation
S72, S76
Incomplete documentation
S34
Insufficient documentation
S27, S38, S72
Lack of code comments
S14, S22, S24, S26, S29, S38, S58, S69, S72, S93
Not specified
S9, S47, S48, S52, S63, S77
Infrastructure TD (3/6)
Old technology in use
S24
Old supporting tools in use
S24
Lack of continuous integration
S69
Lack of automated deployment
S69
Poor release planning
S53
Not specified
S12, S74, S77
Versioning TD (1/1)
Unnecessary code forks
S32
Multi-version support
S32
Defect TD (7/11)
Defects/bugs
S6, S12, S19, S34, S43, S69, S72, S80, S90, S93, S94
Legend
TD
is comprised of
TD type (studied No./total No.)
Under-studied TD sub-type
Studied TD sub-type
is mentioned or investigated in
Related studies

Fig. 8. TD classification tree.

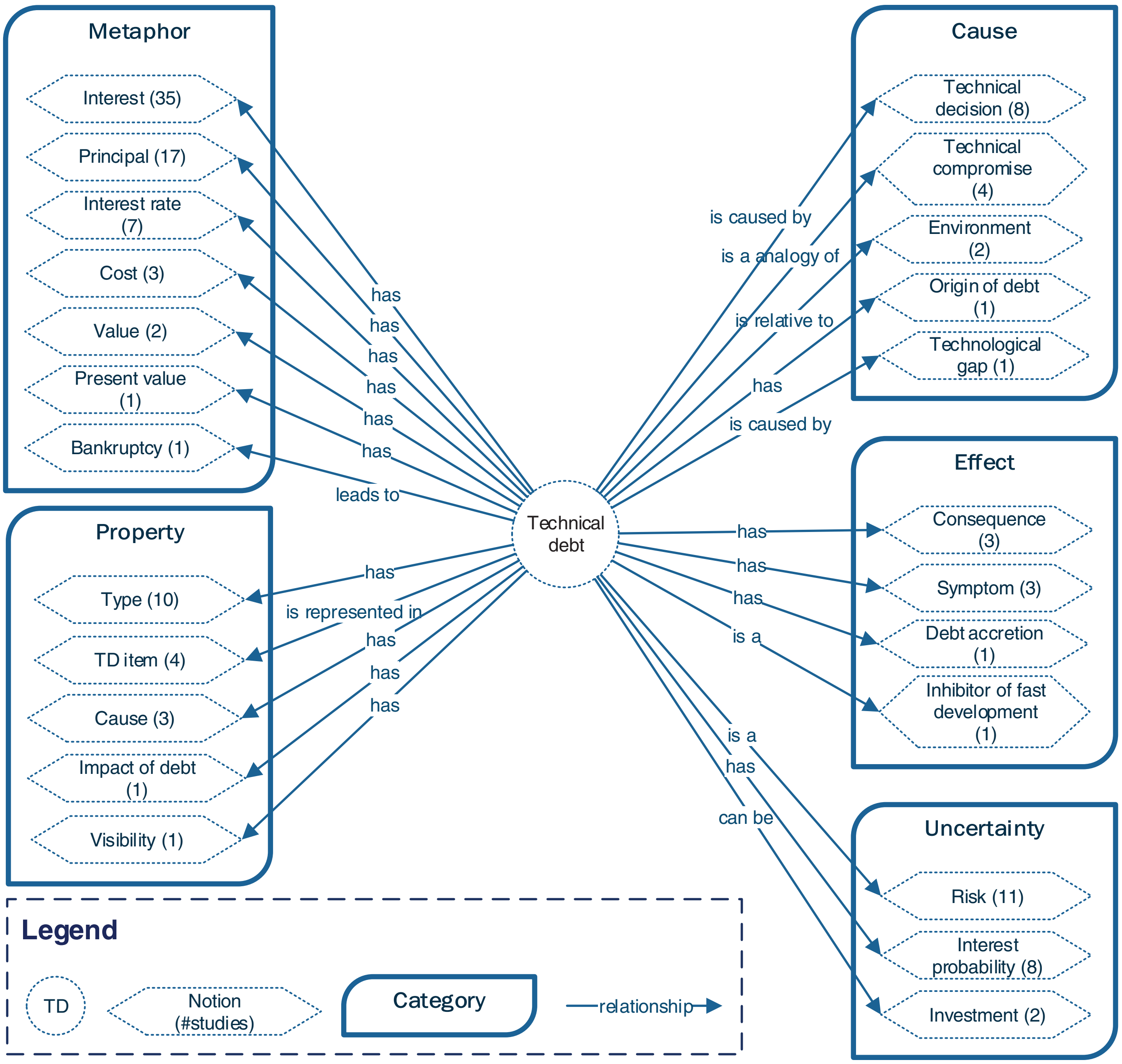

Fig. 9. TD-related notions, their categories, and the number of related studies for each notion.

TD is incurred. Note that some notions could belong to more than one category, but we classified each notion into the primary category which we considered fitting the notion best. For instance, the notions *cost* and *value* can also be classified into the category "Property".

4.4.4. Compromised quality attributes (RQ4)

RQ4: *Which quality attributes are compromised when TD is incurred?*

This subsection presents the quality attributes (QAs) compromised when TD is incurred. When extracting the raw data of quality attributes from the selected studies, we observed that the extracted quality attributes did not comply with a uniform quality model; for example some of these QAs are adopted from the ISO/IEC 9126 standard [2001], and some are defined and used by the authors themselves. All the extracted QAs are about software product quality and none is about quality in use (ISO/IEC, 2011) (i.e., software quality from the perspective of human–computer interaction). Thus, we map the extracted QAs to the product model defined in ISO/IEC 25010 standard [2011], which is the latest revision of ISO/IEC 9126.

Table 6 shows the QAs that are compromised when TD is incurred. Many selected studies only mention the QAs compromised but do not specify the sub-QAs. Since ISO/IEC 25010 does not distinguish between the ease of implementing new requirements and fixing bugs as different QAs, we classified into modifiability the cases where the ability to implement new requirements is negatively influenced.

Most studies argue that TD negatively affects the maintainability (maintainability as a whole or its sub-QAs) of software systems; other QAs and sub-QAs are only mentioned in a handful of studies each.

4.4.5. Limits of the technical debt metaphor (RQ5)

RQ5: *What are the limits of the TD metaphor?*

We only found three studies discussing the limits of the TD metaphor. In S67, Rooney challenges the TD metaphor and states that it is not sufficient to describe the systems built in modern development approaches, such as XP, since the TD incurred in modern approaches is usually paid off in the near future. The accumulated debt is thus relatively little, while the code keeps value over time. In S71, Schmid argues that there are several major shortcomings of the TD metaphor: (1) TD lacks a standard unit of measurement; (2) the amount of interest that needs to be repaid directly depends on the future development that is affected by the TD. Thus, it is

Table 5
TD-related notions and their related studies.

Notion	Description	Related studies
Interest	The extra effort needed to modify the part of the software system that contains TD [S9, S34]	S1, S2, S9, S10, S11, S12, S15, S16, S17, S20, S22, S23, S24, S34, S35, S39, S40, S42, S50, S53, S55, S56, S61, S62, S65, S71, S72, S73, S75, S77, S78, S85, S91, S93, S94
Principal	The estimated cost of resolving a given type of TD [S9]	S9, S11, S12, S16, S17, S24, S34, S35, S39, S40, S55, S72, S73, S77, S85, S93, S94
Risk	TD is a type of risk for a software project, since TD can eventually hurt the health of the project if the TD is left unresolved	S1, S3, S12, S17, S34, S35, S52, S53, S65, S72, S93
Type	TD can be classified into various types, such as architectural TD and documentation TD	S9, S10, S17, S22, S34, S39, S52, S72, S73, S94
Interest probability	"The probability that a particular type of TD will in fact have visible consequences [S9]"	S9, S12, S35, S39, S40, S72, S73, S93
Technical decision	TD is caused by technical decisions, such as architecture decisions and implementation (coding) decisions	S10, S12, S24, S52, S56, S61, S69, S78
Interest rate	The percentage of the interest of a specific piece of TD out of its principal in a given period	S2, S9, S10, S11, S53, S85, S91
TD item	A TD item is a unit of TD in a software system. An example of a TD item is a "God" class with information about its location, estimated cost and benefit of it not being repaid, responsible developer, and its TD type (design TD in this case). The TD in a software system is comprised of a number of TD items	S34, S39, S72, S93
Technical compromise/tradeoff/shortcut	Compromises between optimal technical solutions and sub-optimal ones (that incur TD) in a specific environment	S1, S11, S12, S55
Cost	The cost of incurring TD, i.e., the sum of principal and interest	S11, S17, S34
Cause	The reason for the existence of TD	S30, S42, S52
Symptom	A sign manifesting the existence of TD	S17, S52, S55
Consequence/effects	The influences of incurring TD on the software system	S11, S52, S62
Value/benefit	The potential benefit of incurring TD, i.e., the benefit that can be immediately obtained and the possible future benefit after TD is incurred	S9, S34
Investment	TD can be an investment for a software project as long as the benefit of incurring the TD outweighs its cost	S34, S48
Environment	"In software engineering projects, debt is relative to a given or assumed environment [S9]"	S9, S85
Origin of debt/intentionality	Origin of debt refers to the intentionality of the debt, i.e., whether the TD is incurred intentionally or unintentionally [S9]	S9, S93
Impact of debt	The scope of the influence when changing the system to repay TD, e.g., the number of affected components [S9]	S9
Inhibitor of fast development	Extra effort needs to be spent in working on and around the software part containing the unsolved TD, which negatively affects the speed of development	S45
Visibility	TD should be visible to stakeholders, particularly to the system-level decision-makers and the people who will eventually repay it [S9]	S9
Present value	The immediate benefit obtained after TD is incurred [S9]	S9
Debt accretion	TD may increase non-linearly and accruing too much debt may lead to a software system into a serious, even irreparable, state [S9]	S9
Bankruptcy	Bankruptcy happens when the part of the software system which contains TD is no longer viable to support future development and a complete rewrite and a new platform are needed [S63]	S63
Technological gap	Gaps in technology caused by the invisible aspects of natural software aging and evolution, such as technological obsolescence, change of environment, rapid commercial success, advent of new and better technologies, and so forth [S47]	S47

impossible to relate TD directly to an interest rate or an interest period. In S1, Allman thought that people who take on TD are usually not the ones who repay the TD. This may encourage people to take more TD to accelerate their development.

4.5. TD management

This section answers the research questions on TD management (TDM), i.e., RQ6, RQ7, RQ8, and RQ9.

4.5.1. TDM activities (RQ6)

RQ6: *What are the different activities of TDM?*

TDM is composed of a set of activities that prevent potential TD from being incurred or deal with existing TD to keep it under a reasonable level. Table 7 shows nine TDM activities that are collected and refined from the selected studies. Detailed explanations of these TDM activities are presented below:

- **TD identification** detects TD caused by intentional or unintentional technical decisions in a software system through specific techniques, such as static code analysis.
- **TD measurement** quantifies the benefit and cost of known TD in a software system through estimation techniques, or estimates the level of the overall TD in a system.
- **TD prioritization** ranks identified TD according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases.
- **TD prevention** aims to prevent potential TD from being incurred.
- **TD monitoring** watches the changes of the cost and benefit of unresolved TD over time.

Table 6
QAs from ISO/IEC 25010 compromised by incurring TD.

QA Sub-QA	Studies
Functional suitability	None
Functional completeness	None
Functional correctness	S85, S92
Functional appropriateness	None
Performance efficiency	S14, S17, S50
Time behavior	None
Resource utilization	None
Capacity	None
Compatibility	S68
Co-existence	None
Interoperability	S45, S85
Usability	S93
Appropriateness recognizability	None
Learnability	None
Operability	S85
User error protection	None
User interface aesthetics	S93
Accessibility	None
Reliability	S14, S38, S50, S51
Maturity	None
Availability	None
Fault tolerance	S17
Recoverability	S17
Security	S14, S17, S50, S79
Confidentiality	None
Integrity	None
Non-repudiation	None
Accountability	None
Authenticity	None
Maintainability	S1, S3, S7, S9, S10, S11, S14, S15, S20, S21, S25, S29, S34, S35, S36, S37, S38, S41, S42, S44, S46, S47, S49, S50, S51, S52, S55, S57, S60, S62, S72, S73, S74, S76, S78, S79, S81, S82, S86, S87, S88, S89, S90, S91, S92, S94
Modularity	S66
Reusability	S14, S50, S68, S86
Analyzability	S17, S62
Modifiability	S1, S3, S5, S9, S11, S14, S17, S18, S31, S38, S45, S47, S50, S51, S57, S60, S61, S62, S71, S78, S85
Testability	S14, S50, S51, S62, S85
Portability	S14, S21, S50, S76
Adaptability	S60, S79, S85
Installability	None
Replaceability	None

Note: Some studies mentioned both main QAs and sub-QAs, so we classified these studies into both the main QAs and applicable sub-QAs. Other studies only mentioned sub-QAs or the main QAs without specifying sub-QAs.

Table 7
TDM activities and the numbers of related studies.

TDM activity	No. of studies	%
TD repayment	59	63
TD identification	51	54
TD measurement	49	52
TD monitoring	19	20
TD prioritization	17	18
TD communication	17	18
TD prevention	9	10
TD representation/documentation	4	4

- **TD repayment** resolves or mitigates TD in a software system by techniques such as reengineering and refactoring.
- **TD representation/documentation** provides a way to represent and codify TD in a uniform manner addressing the concerns of particular stakeholders.
- **TD communication** makes identified TD visible to stakeholders so that it can be discussed and further managed.

The TDM activities mentioned above have received significantly different levels of attention, considering the number of selected studies that have mentioned or deeply investigated each TDM activity. As shown in Table 7, TD repayment, identification, and measurement have been mentioned or investigated in more than half of the total selected studies, with percentages 63% (59 out of 94), 54% (51 out of 94), and 52% (49 out of 94), respectively. TD representation/documentation have received the least attention with only four (4%) related selected studies.

4.5.2. TDM approaches (RQ7)

RQ7: *What approaches are used in each TDM activity?*

We present TDM approaches in two perspectives: (1) distribution of studies over TDM activities, TD types, and publication years, and (2) categories of approaches for each TDM activity. The names of the approach categories are indicative and other names could also be used, e.g., the category "Code metrics" could also be called "Code measurement".

4.5.2.1. Primary studies of TDM. Fig. 10 shows the distribution of the 58 selected studies, which mention, propose, or use one or more TDM approaches, over TDM activities, TD types, and publication years. A bubble in the left part of the figure represents one or more studies that include approaches for a TDM activity published in a certain year. For instance, the left-bottom bubble in Fig. 10 denotes that the seven studies (S39, S40, S57, S71, S88, S93, and S94) mentioned, proposed, or used approaches for TD identification and these studies were published in 2013. In the right part of Fig. 10, a bubble denotes one or more studies include approaches for a TDM activity dealing with a specific type of TD. Note that, we use "general TD" as a TD type in Fig. 10 when an approach is not dedicated to handling a specific TD type (one of the 10 TD types in Fig. 8). During 1992–2008, only one study was published per year in 1992, 2000, 2006, and 2007, two studies were published in 2009, and no study was published in the other years. Most of the studies (90%, 52 out of 58) on TDM approaches were published between 2010 and 2013.

Fig. 11 shows the distribution of studies where one or more concrete approaches were mentioned, used, or proposed for TDM activities. There are significantly more studies that presented approaches for TD identification, measurement, and repayment than for the rest of the TDM activities. Note that the numbers in Fig. 11 are different than the numbers in Table 7, as the former concerns studies that include approaches for the different TD activities, while the latter may simply mention TDM activities without discussing approaches.

Fig. 12 shows the distribution of studies where one or more concrete TDM approaches were mentioned, used, or proposed to manage various TD types. There are significantly more studies that present TDM approaches to deal with code TD, design TD , and architectural TD. There is no study including concrete TDM approaches for documentation TD and versioning TD. As shown in Fig. 8, documentation TD is studied in 10 studies, which is not conflicting with the fact of no study including concrete TDM approaches for documentation TD. The reason is that the 10 studies only discuss documentation TD (e.g., on its consequences and causes), or have empirical studies on documentation TD but without specify what TDM approaches were used in the empirical studies.

4.5.2.2. TD identification. TD identification approaches collected from 26 studies are categorized into four categories as shown in Table 8. Code analysis and dependency analysis are two most popular approaches.

4.5.2.3. TD measurement. TD measurement approaches collected from 24 studies are classified into six categories as shown in Table 9. Calculation model is the most used category for TD measurement.

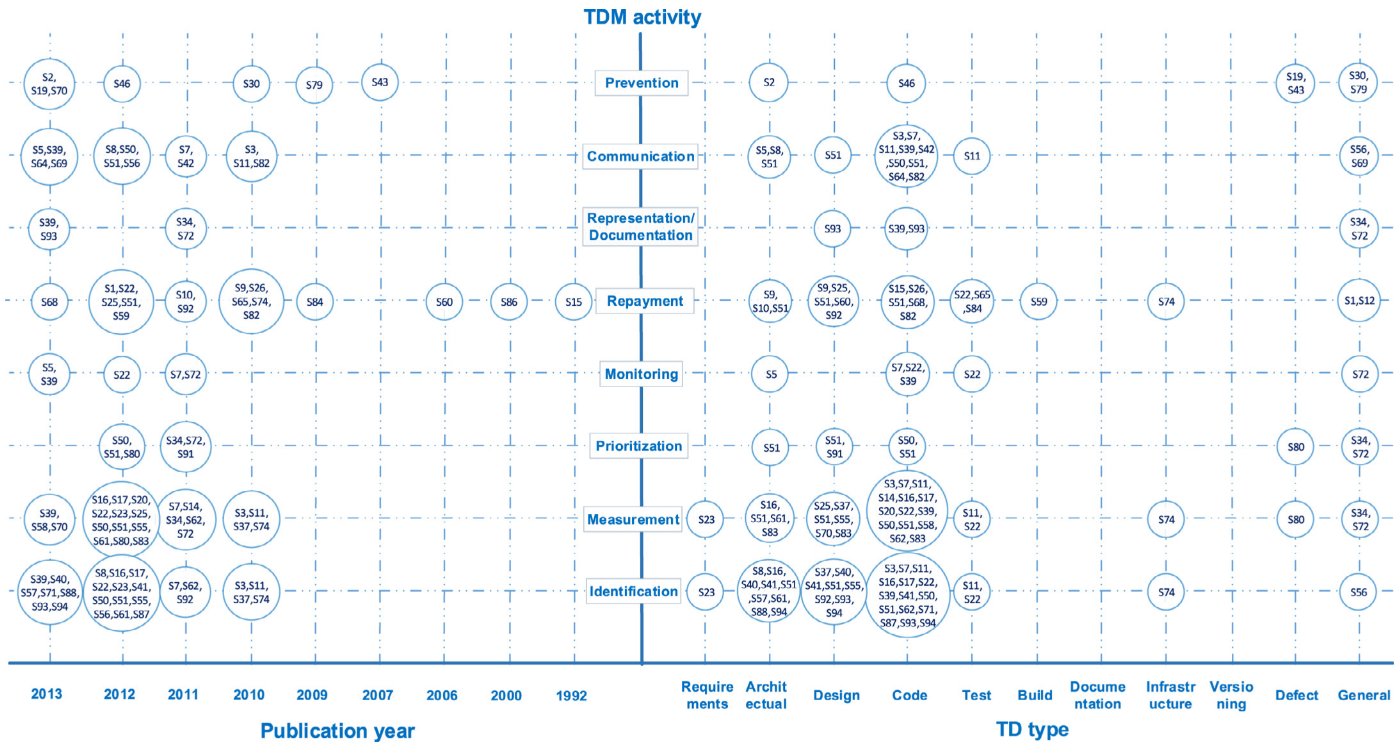

Fig. 10. Distribution of studies on TDM approaches over TDM activities, TD types, and publication years.

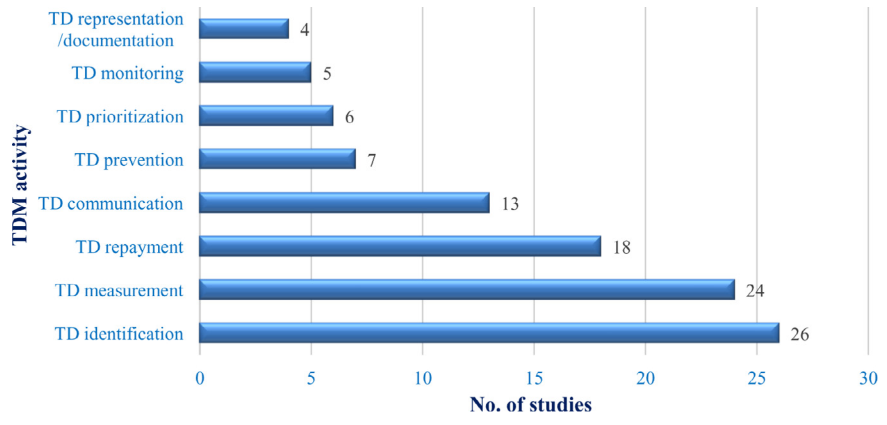

Fig. 11. Distribution of studies on TDM approaches over TDM activities.

Table 8
TD identification approaches and related studies.

Category	Description	Studies
Code analysis	Analyze source code to identify violations of coding rules, lack of tests; calculate software metrics based on source code to identify design or architecture issues	S3, S7, S11, S16, S17, S22, S37, S39, S40, S41, S50, S51, S55, S62, S71, S87, S92, S93, S94
Dependency analysis	Analyze dependencies between different types of software elements (e.g., components, modules)	S8, S51, S56, S57, S88, S94
Check list	Check against a list of predefined scenarios where TD is incurred [S74]	S74
Solution comparison	Compare the actual solution with the optimal solution in some dimension, such as cost/benefit ratio. If the actual solution is not the optimal one, then TD is incurred	S23, S61

4.5.2.4. TD representation/documentation. Four studies (S34, S39, S72, and S93) propose an approach to represent TD, and all four of them perform the representation with TD items. A TD item is a unit of TD of a software system, for example a "God class" (see Table 5). However, the proposed formats of TD items are different. The fields of a TD item are shown in Table 10, in which we can see that the fields *ID*, *Location*, *Responsible/author*, *Type*, and *Description* are included in a TD item in all four studies.

4.5.2.5. TD prioritization. Six studies explicitly discuss concrete approaches for TD prioritization, which are categorized into four categories as shown in Table 11.

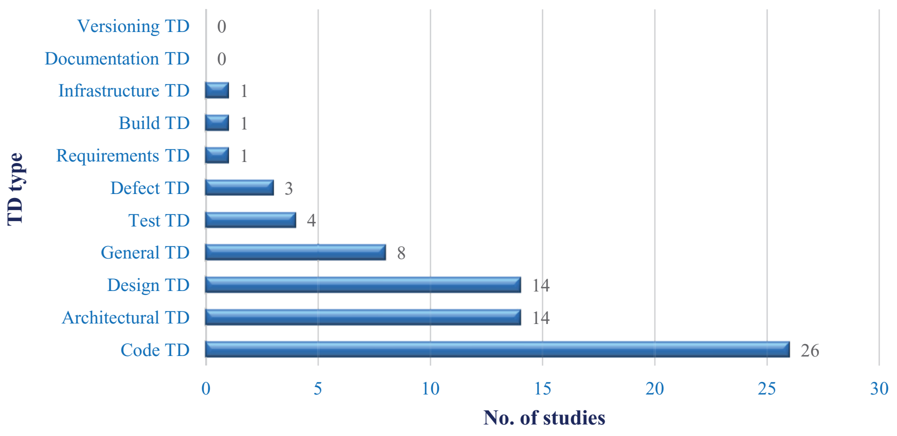

Fig. 12. Distribution of studies on TDM approaches over TD types.

Table 9
TD measurement approaches and related studies.

Category	Description	Studies
Calculation model	Calculate TD through mathematical formulas or models	S11, S14, S16, S17, S20, S22, S37, S39, S50, S51, S55, S58, S61, S62, S70, S72, S83
Code metrics	Calculate TD using metrics of source code	S3, S7, S25, S39, S83
Human estimation	Estimate TD according to experience and expertise	S34, S72
Cost categorization	Estimate various types of the cost of handling the incurred TD [S80]	S80
Operational metrics	Indicate TD using quality metrics of product operation [S74]	S74
Solution comparison	Calculate the distance between the actual solution and the optimal solution [S23]	S23

Table 10
Fields of a TD item and related studies.

Field	Description	Studies
ID	A unique identifier for a TD item	S34, S39, S72, S93
Location	The location of the identified TD item	S34, S39, S72, S93
Responsible/author	The person who is responsible for repaying the TD item	S34, S39, S72, S93
Type	The TD type that this TD item is classified into, e.g., architectural TD	S34, S39, S72, S93
Description	General information on the TD item	S34, S39, S72, S93
Date/time	The date or time when the TD item is identified	S34, S39, S72
Principal	The estimated cost of repaying the TD item	S34, S72, S93
Interest amount	The estimated extra cost of tolerating the TD item	S34, S72, S93
Interest probability	The probability that the interest of this TD item needs to be repaid	S72, S93
Interest standard deviation	The estimated difference between the estimated interest amount and the actual (future) interest amount	S34
Correlations with other debt items	Relationships between this TD item and other TD items	S34
Name	The name of a specific type of TD in a TD item (in S39, a TD item may include multiple TD types)	S39
Context	A certain implementation context (e.g., programming language used) of a specific TD type in a TD item.	S39
Propagation rules	How this TD item impacts the related parts of the software system	S39
Intentionality	Intentionally or unintentionally incurred	S93

4.5.2.6. TD monitoring. Five studies explicitly discuss concrete approaches for TD monitoring, which are shown in Table 12.

4.5.2.7. TD repayment. TD repayment approaches collected from 18 studies are classified into seven categories as shown in Table 13. Among these categories of TD repayment approaches, refactoring is mostly used.

4.5.2.8. TD communication. TD communication approaches collected from 13 studies are categorized into six categories as shown in Table 14.

4.5.2.9. TD prevention. TD prevention approaches collected from 7 studies are classified into four categories as shown in Table 15.

4.5.3. *TDM tools (RQ8)*

RQ8: *What tools are used in TDM and what TDM activities are supported by these tools?*

Tools are important for performing TDM activities, thus, we examined the tools that are mentioned, used, or developed in the selected studies for the purpose of managing TD. Table 16 shows the functionalities provided, TD types handled, artifacts taken as input, and the TDM activities supported by the tools. In addition, the column "Vendor" denotes the organizations that develop the tools; the column "Study" lists the studies that mentioned, used, or developed the tools in the corresponding rows; the column "TD type" describes the TD types that are dealt with by the tools; the column "Artifact" describes the input of the tools; and the column "Free of charge" denotes whether the tool can be used for free. If a study discusses both TDM approaches and tools, this study is mapped to both Fig. 10 and Table 16.

Table 11
TD prioritization approaches and related studies.

Category	Description	Study
Cost/benefit analysis	If resolving a TD item can yield a higher benefit than cost, then this TD item should be repaid. TD items with higher cost/benefit ratios of repayment should be repaid first [S91]	S80, S91
High remediation cost first	TD items that are more costly to resolve should be repaid first [S50, S51]	S50, S51
Portfolio approach	The portfolio approach considers TD items along with other new functionalities and bugs as risks and investment opportunities (i.e., assets). "The goal of portfolio management is to select the asset set that can maximize the return on investment or minimize the investment risk [S34]"	S34
High interest first	TD items incurring higher interest should be repaid first [S72]	S72

Table 12
TD monitoring approaches and related studies.

Category	Description	Study
Threshold-based approach	Define thresholds for TD related quality metrics, and issue warnings if the thresholds are not met [S22]	S22
TD propagation tracking	Track the influences of TD through dependencies between other parts of a system and the parts of the system that contains TD [S39]	S39
Planned check	Regularly measure identified TD and track the change of the TD [S7]	S7
TD monitoring with quality attribute focus	Monitor the change of quality attributes that detrimental to TD, such as stability [S5]	S5
TD plot	Plot various aggregated measures of TD over time and look at the shape of the curve to observe the trends [S72]	S72

Table 13
TD repayment approaches and related studies.

Category	Description	Study
Refactoring	Make changes to the code, design, or architecture of a software system without altering the external behaviors of the software system, in order to improve the internal quality [S10]	S1, S9, S10, S12, S22, S25, S51, S60, S68, S86, S92
Rewriting	Rewrite the code that contains TD	S1, S10, S15, S51
Automation	Automate manually-repeated work, e.g., manual tests, manual builds [S84], and manual deployment [S74]	S50, S74, S84
Reengineering	Evolve existing software to exhibit new behaviors, features, and operational quality	S9, S10, S12
Repackaging	Group cohesive modules with manageable dependencies to simplify the code [S12]	S12
Bug fixing	Resolve known bugs [S26]	S26
Fault tolerance	Strategically place runtime exceptions where the TD is [S82]	S82

Table 14
TD communication approaches and related studies.

Category	Description	Study
TD dashboard	A dashboard displays TD items, types, and amounts in order to get all stakeholders informed of the existence of the TD	S3, S50, S51, S69
Backlog	All identified TD items as well as anything to be resolved in the development are put into the backlog of the software project, so that the TD items can be treated as important as known bugs and unimplemented planned features and functionalities	S11, S64, S82
Dependency visualization	Visualize the undesirable dependencies (e.g., overly complex dependencies) between software elements (e.g., components and packages)	S8, S56
Code metrics visualization	Visualize code metrics in some tools such as code maps and highlight those software elements with bad measured quality (e.g., code complexity)	S7, S42
TD list	A TD list keeps all identified TD items and make them visible to stakeholders	S5, S39
TD propagation visualization	Show the connections between different TD items, and how a TD item affects and is affected by other TD items [S39]	S39

Table 15
TD prevention approaches and related studies.

Category	Description	Study
Development process improvement	Improve current development processes to prevent the occurrences of certain types of TD [S19, S46]	S19, S43, S46
Architecture decision making support	Evaluate potential TD caused by different architecture design options, and then choose the option with less potential TD [S70]	S2, S70
Lifecycle cost planning	Develop cost-effective plans that look at the system throughout the lifecycle to minimize overall TD of the system [S79]	S79
Human factors analysis	Cultivate a culture that minimizes the unintentional TD caused by human factors, e.g., indifference and ignorance [S30]	S30

As shown in Table 16, most (86%, 25 out of 29) of the tools take source code as input, only one (T1) takes .NET Assemblies as input, one (T15) takes requirements and solutions as input, and one (T27) takes compiled binaries as input. As shown in Table 16, except for the four dedicated TDM tools (denoted with $*$), the rest of the collected tools support no more than two TDM activities. As shown in Fig. 13, TD identification is widely supported by 86% (25 out of 29) of the tools; TD measurement and communication also received support by 28% (8 out of 29) and 17% (5 out of 29) of the tools respectively, but that is much less than TD identification. The rest five TDM activities are supported by very few tools; especially TD prevention has no supporting tool. Only four tools are dedicated to managing TD, while

Table 16
Tool support for TDM activities.[a]

#	Tool	Functionality	Vendor	Study	TD type	Artifact	TD identification	TD measurement	TD prioritization	TD monitoring	TD repayment	TD representation/ documentation	TD communication	TD prevention	Free of charge
T1	SIG Software Analysis Toolkit	Calculating code properties	SIG	S62	Code TD	Code	X	X							X
T2	Google CodePro Analytix	Calculating code metrics	Google	S25	Design TD: code smells	Code		X							X
T3	iPlasma	Calculating code metrics	Politehnica University of imisoara	S25	Design TD: code smells	Code		X							X
T4	Eclipse Metrics	Calculating code metrics	State of Flow	S25	Design TD: code smells	Code		X							X
T5	Rational AppScan	Identifying security flaws in source code	IBM Rational	S3	Code TD	Code	X								
T6	PMD	Looking for potential problems in source code	OSS	S3	Code TD	Code	X								X
T7	PHPMD	Detecting mess (e.g., overcomplicated expressions) in PHP code	OSS	S3	Code TD	Code	X								X
T8	NDepend	Calculating .NET code metrics	SMACCHIA.COM SARL	S3	Code TD	Code	X								
T9	NCover	Analyzing code coverage for .NET	NCover.com	S3	Test TD	Code	X								
T10	FxCop	Analyzing managed code assemblies to identify compliance issues against .NET programming guidelines	Microsoft	S3	Code TD	.NET assemblies	X								X
T11	CodeXpert	Automating PL/SQL code quality and standards reviews	Dell	S3	Code TD	Code	X								X
T12	Cobertura	Analyzing code coverage for Java	OSS	S3	Test TD	Code	X								X
T13	Checkstyle	Checking Java code against coding standards	OSS	S3	Code TD	Code	X								X
T14	Software maps tool*	Visualizing code quality of source code files	University of Potsdam	S7	Code TD	Code	X	X		X			X		U
T15	RE-KOMBINE	Identifying and measuring requirements TD	University of British Columbia	S23	Requirements TD	Requirements, solutions	X	X							U
T16	Code Christmas Trees	Visualizing code complexity and coverage	Centric Consulting	S42	Code TD	Code							X		U
T17	CAST's Software's Applications Intelligence Platform	Identifying violations in source code and categorizing the violations by quality attributes	CAST	S16, S17	Code TD, architectural TD	Code	X								
T18	Technical Debt Evaluation (SQALE) plugin for SonarQube*	Analyzing, measuring, visualizing, and prioritizing TD based on SQALE quality model	Inspearit	S50, S51	Code TD	Code	X	X	X				X		
T19	STAN	Calculating the structure quality metrics of Java systems	Odysseus Software GmbH	S37	Design TD	Code		X							
T20	Resource Standard Metrics	Calculating source code metrics and analyzing code quality to find style violations and logic problems	M Squared Technologies LLC	S37	Design TD, Code TD: coding violations	Code	X	X							

(continued on next page)

Table 16 (*continued*)

#	Tool	Functionality	Vendor	Study	TD type	Artifact	TD identification	TD measurement	TD prioritization	TD monitoring	TD repayment	TD representation/ documentation	TD communication	TD prevention	Free of charge
T21	DebtFlag∗	Supporting TDM by maintaining an implementation level representation of TD and providing needed information for project level management; providing an Eclipse Plugin to capture TD by using lightweight documentation tool, and a Web application to manage TD.	University of Turku	S39	Code TD	Code	X			X		X	X		U
T22	RBML compliance checker	Calculating a distance between a realization of a design pattern and the intended design	Montana State University	S40	Design TD: grimes	UML models	X								X
T23	A tool to identify bad dependencies	Identifying bad intra- and inter- module dependencies	University of Waterloo	S88	Architectural TD, Code TD	Code	X								U
T24	Sonar TD plugin∗	Identifying and measuring TD in the form of low code coverage, design, violations, complexity, comments	SonarSource SA	S3, S11, S26, S58, S69, S83	Code TD, Test TD, Documenta- tion TD	Code	X	X	X	X	X		X		X
T25	SonarQube	Open platform for managing code quality	SonarSource SA	S3	Code TD	Code	X								X
T26	SonarQube COBOL Plugin	Performing objective and automated COBOL code reviews against coding best practices	SonarSource SA	S3	Code TD: coding violations	Code	X								
T27	CLIO	Identifying modularity violations	Drexel University	S40, S41, S94	Architectural TD: architectural compliance issues	Compiled binaries	X								U
T28	CodeVizard	Identifying code smells	University of Maryland	S41, S40, S93, S94	Design TD: code smells	Code	X								U
T29	FindBugs	Identifying automatic static analysis issues	University of Maryland	S3, S41, S87, S93, S94	Code TD: coding violations	Code	X								X

[a] "U" in column *Free of charge* denotes unknown; "∗" in column *Tool* denotes dedicated tools specifically built for TDM; "OSS" in column *Vendor* denotes the tool developed in OSS (Open Source Software) community.

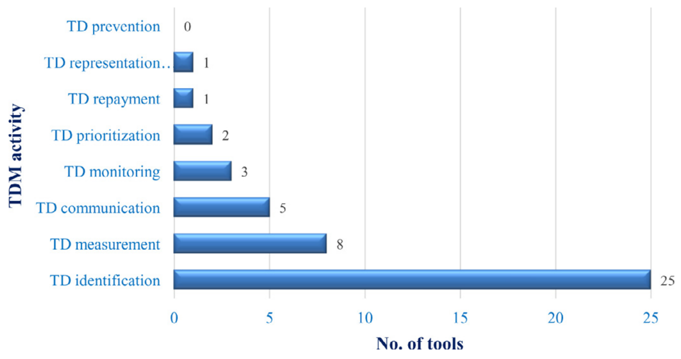

Fig. 13. Number of tools supporting TDM activities.

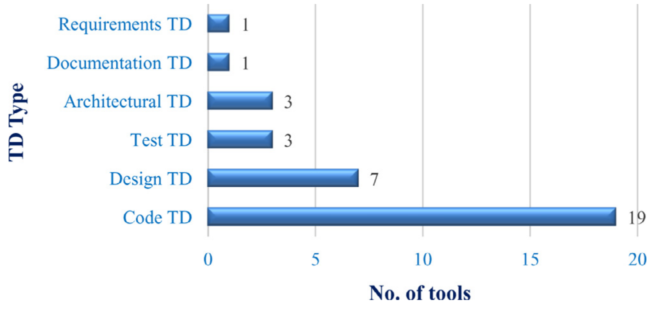

Fig. 14. Number of tools supporting TD types.

most of the collected tools in Table 16 are borrowed from other fields (e.g., code quality analysis). Most of the tools deal with code TD and design TD (see Fig. 14).

4.5.4. Challenges in TDM (RQ9)

RQ9: *What challenges for TDM have been identified?*

We identified a number of challenges explicitly stated in the selected studies.

- *Challenges in managing induced and unintentional TD*. S44 argues that TD can be induced by stakeholders other than architects in the project or across the portfolio. For example such TD can be caused by imposing a strict release date at the cost of software quality, or through the cascaded effect from decisions made on other projects on which a given project depends. Such TD is beyond the control of the development team of a given project, thus is hard to manage. S44 also argues that it is difficult to deal with the unintentional TD caused by situations such as acquisition, new alignment requirements, or changes in the market ecosystem.
- *Difficulties in business and economic value transformation*. S86 holds that it is very hard for organizations to assign business value to the intrinsic quality (i.e., compromised QAs presented in Section 4.4.4 when TD is incurred) of the software product itself. S24 holds that it is hard to translate TD into economic consequences.
- *Challenges in TD measurement*. S1 believes that it can be difficult to know in advance which TD items will ultimately have the highest cost. S91 argues that, in many projects, the cost and benefit of refactoring (an approach to repaying TD) cannot be easily quantified and estimated. S52 holds that measuring TD is not easy because its impacts are not uniform. S61 argues that a key challenge in iterative development is the ability to quantify the value of infrastructure and quality-related tasks, which quite often are architectural. S5 argues that, due to the lack of TD measures, development teams could not make a strong case to the business side to convince business stakeholders to invest in fixing technical shortcuts taken. This often leads to a disruptive bug-fixing iteration or major redesign.
- *Challenges in TD prevention*. S19 argues that in agile development it is challenging to reduce the number of defects and at the same time improve non-functional quality (to prevent incurring TD).
- *Lack of an underlying theory and models to aid TD identification and measurement*. S72 argues that a comprehensive TD theory should be developed to formalize the relationship between the cost and benefit of the TD concept, and subsequently practical TDM approaches should be developed and validated to exploit the TD theory in management decision making.

5. Discussion

This section presents our interpretation of the results of this systematic mapping study and the implications of the results for researchers and practitioners who are working on TD.

5.1. Technical debt concept

This subsection discusses the classification of TD, conflicting opinions on TD, QAs compromised when TD is incurred, and the current state of the research of TD.

5.1.1. Classification of TD

In the data synthesis phase, we encountered some difficulties in TD classification, as in several cases the TD type was not clear from the study itself or was conflicting with other studies or literature. We discuss how we dealt with them below.

(1) In S55, the author argues that duplicate code is design debt, but we classified it into code TD since code duplication affects a fragment of a function or method.
(2) S85 argues that deferred functionality is TD, but we did not include it in the TD classification tree, to be consistent with the bulk of TD literature which does not consider this as TD.
(3) Some primary studies only mention that quality issues in code are TD, without specifying details of the quality issues. For example, in S34 the authors deem that "low quality code" is TD, in S28 the authors consider "code quality deficits" are TD, and in S15 the author argues that "not-quite-right code" is TD. We classified these kinds of quality issues in code as low-quality code, which is a sub-type of code TD.
(4) We did not take structural debt (in S47) as a TD category since structural issues can be merged into architectural TD or design TD depending on their granularity. System-level structural quality issues can be regarded as a kind of architectural TD. Class- and method-level structural quality issues can be seen as a kind of design TD.
(5) The sub-type "bad dependency" of build TD is different from undesired dependencies in system-level structure issues of architectural TD. The former refers to dependencies on external components and libraries that are frequently changed while the latter are dependencies on internal components.
(6) In S71, the authors mention that sub-optimal structured code is TD without giving further details. We classified the sub-optimal structured code into complex code, a sub-type of code TD.
(7) S83 investigates portfolio debt, but does not give a definition for it. Hence, we did not include it as a TD type.
(8) S93 introduces usability debt as the lack of a common user interface template. We included this case into design TD, as user interface design is part of design.
(9) Some primary studies (e.g., S14) mentioned quality debt. In fact, any TD hurts one or more quality attributes (or QAs are compromised when TD is incurred). We consider quality debt too general, thus did not include it in the TD classification tree.

We found that researchers and practitioners tend to use the term "debt" arbitrarily, without attaching a clear and precise meaning to it;

this leads to ambiguous interpretation of the term. They also tend to connect any software quality issues to debt, such as code smells debt, dependency debt, and usability debt. It seems that the term "debt" is a buzzword that is being used very often, leading to ambiguous interpretation of its meaning; we hope that this systematic mapping study on TD will provide sufficient clarification of the TD concept.

5.1.2. *Conflicting opinions on TD*

The concept TD was coined to describe the technical shortcuts taken in coding in order to speed up the development and to meet an urgent release deadline (Cunningham, 1992). TD was originally used to describe 'not-quite-right code'. In recent years, TD has been extended to other phases in the software development lifecycle and more and more concepts were put under the umbrella of TD (Kruchten et al., 2013). In this mapping study, we found some conflicting opinions on the scope of TD.

- Regarding the scope of TD, Barton and Sterling deem that TD concerns issues found in the code that affect future development (Barton and Sterling, 2010), while many others such as Brown et al. (2010) hold that TD is a broader concept concerning also issues in other phases of the lifecycle, e.g., requirements or architecture, which harm the long-term health of a software project.
- In S85, Theodoropoulos et al. argue that deferred functionality is TD, while S17, S47, S48, and S64 explicitly point out that unimplemented features and functionalities are not TD.
- Eleven studies consider defects as TD (see Fig. 8), while four studies (S17, S47, S48, and S64) explicitly emphasize that defects should not be included into TD.
- Trivial code quality issues are not considered as TD in S4. This point actually contradicts the TD literature, which takes into account trivial code quality issues. However, the authors of S4 do not define what trivial code quality issues are, but they take *messy code* as an example. Messy code actually covers code violating coding rules, which is a sub-type (coding violations) of code TD; 17 studies consider coding violations are TD.

5.1.3. *Quality attribute compromised*

As shown in Table 6, most of the selected studies argue that maintainability or its sub-QAs are compromised when TD is incurred. This is in accordance with the original description of TD by Ward Cunningham (Cunningham, 1992). However, several other studies do mention other QAs and sub-QAs that get compromised because of TD (see Section 4.4.4), so maintainability is not the only QA suffering from TD.

5.1.4. *Current research on TD*

Both academia and industry pay significant attention to the research on TD, judging from the numbers of the selected studies conducted by academic researchers and industrial practitioners (40 studies by researchers only, 38 studies by practitioners only, and the rest 16 studies are joint effort from both researchers and practitioners). This is certainly a very encouraging sign, where a software engineering research topic receives a balanced attention by both communities. One potential reason is that the concept of TD originated from industrial practices (Cunningham, 1992) and was first becoming popular in the agile development community. As described in (Tom et al., 2013), there had been a lot of web blogs and online articles discussing TD before it became a popular research topic in academia. During the study selection and data extraction stages of this mapping study, we found that "technical debt" has been a widely-accepted and broadly-used term in the agile development community (see Sutherland et al., 2009; Martin et al., 2009; Birkeland, 2010; Uy and Ioannou, 2008).

Interest and principal are the two most used notions to describe and explain the concept of TD. This is understandable since interest and principal are the basic parts of financial debt, so they can be effectively borrowed to describe and explain the TD concept.

TD is considered as risk in eleven studies and as investment in two studies. Since TD means sacrificed software internal quality, it is always a risk for the future software development. TD has both benefit and cost. When the benefit of TD outweighs its cost, we can strategically incur TD as making an investment. However, when the benefit of TD is less than its cost, incurring TD is riskier. The measurement of the cost and benefit of TD is an important and interesting topic for further research.

The TD community has spent a lot of effort to investigate different types of TD, but spent little effort in distinguishing between TD and non-TD. Only 10 studies explicitly mentioned what should not be included as TD. Studies S17, S47, S48, and S64 argue that defects should not be TD; studies S3, S28, S47, and S48 hold that unimplemented features and functionalities should not be TD; studies S85 considers that lack of supporting processes (e.g., business continuity plan) is not TD; S48 argues that unfinished tasks in the development process are not TD; study S4 points out that trivial code quality issues (e.g., messy code) are not TD; and study S47 considers that low external quality (e.g., usability) is not TD.

Most of the studies concern code TD. This is mainly because dealing with source code means there are several available tools supporting the identification, measurement, and repayment of this type of TD. Another possible reason is that code TD is concrete and easy to understand. Finally, practitioners are working on code every day and they already have much experience in some TDM activities (e.g., TD identification and TD repayment) and approaches (e.g., code analysis with tools and code refactoring).

Test TD is the second most studied TD type. The main reason is probably because many tools can analyze code test coverage and automated unit tests. Significant effort also has been spent in architectural TD, partly because the system-level structure quality issues can be identified and measured by dependency analysis based on source code. Design TD is also investigated in many selected studies. Most effort on design TD research has been spent in code smells, one particular sub-type of design TD. The potential reason is that code smells are well recognized and substantial work has been carried out in both academia and industry (Marinescu, 2012; Fowler et al., 1999).

Requirements TD, build TD, infrastructure TD, and versioning TD received little attention from the software engineering community. One potential reason is that requirements TD is more abstract, also concerns business instead of purely technical issues, and is not so well defined as other types of TD. Regarding build TD, infrastructure TD, and versioning TD, these do not directly influence software product quality, but impact the productivity of software development; thus, there may not be much awareness regarding these TD types in their respective communities.

5.2. *Technical debt management*

5.2.1. *TDM activities*

We distilled eight TDM activities from the extracted data of selected studies. TD repayment, identification, and measurement are the three activities that gained the most attention from the software engineering community. These three TDM activities are the most fundamental activities in TDM, as they correspond to three fundamental questions on TD: Where is the TD located? How much is the TD? How to repay the TD?

TD representation/documentation received little attention. One potential reason is that most studies investigating TDM in depth use tools to automatically identify TD based on source code, which is more convenient to track and monitor TD than documented TD, and consequently developers tend not to explicitly document the TD; when the stakeholders need to browse the TD, they just run the tools once again. If TD identification is based on architecture design models, decisions, or other non-code artifacts, TD should be represented in an appropriate form and documented for further management.

5.2.2. TDM approaches

As shown in Fig. 11, TDM approaches are mentioned, used, or proposed mostly for identification, measurement, and repayment. The main reasons are: (1) most of these approaches are either reusing or based on existing approaches and supporting tools in other areas of software development, hence usually less effort is required to propose, use, and validate such approaches than completely new approaches; (2) TD identification, measurement, and repayment are the three fundamental TDM activities (see previous subsection), which urgently need concrete approaches to be applied in real projects.

Code analysis is the main approach of TD identification, since most TD identification approaches reuse existing code analysis tools to detect issues such as coding violations, low test coverage, complex code, code smells and code duplication, but there are few approaches proposed to identify TD.

Calculation models are the most used approach to measure TD. Most calculation models for measuring TD aggregate a set of software metrics that can be calculated based on source code, which is more concrete (close to the running system) for measuring TD than other software artifacts (e.g., UML models). Refactoring is the most used approach for TD repayment in selected studies, since most selected studies on TD repayment are about code and design TD, most of which can be repaid by refactoring.

5.2.3. TDM tools

As shown in Table 16, except for the four dedicated TDM tools, the rest of the collected tools support no more than two TDM activities. Although some tools that are not devoted to TDM can be used to support one or two TDM activities, they can only provide limited support for TDM in the software development process. This implies that more devoted TDM tools need to be developed in order to support more TDM activities for daily use.

Most identified TDM tools as shown in Fig. 13, facilitate TD identification, while other TDM activities gain little support by the tools. Since other TDM activities such as TD prioritization are frequently performed in practice, they also need tool support. Although many tools have been mentioned, used, or developed for TD identification in Fig. 13, there are many more tools for identifying issues in code that have not been presented in the selected studies, since there are many such tools for different programming languages that do not explicitly aim at TDM.

Although there is only one tool in Table 16 that can facilitate TD repayment, many modern Integrated Development Environments (IDEs) provide functionalities supporting code refactoring (TD repayment), such as Microsoft Visual Studio, Eclipse for Java, and plugins for popular IDEs. There is no tool for TD prevention. The potential reason is that TD prevention can be supported mainly by software development process improvement. For instance, continuous integration adopted in the software development process requires a high coverage of automated unit and integration tests; in this way, test TD can be significantly prevented.

Most TDM tools collected in the selected studies deal with code TD and design TD, while other types of TD get little support by the tools. Compared with code TD and design TD, other types of TD (e.g., architectural TD) are either more abstract or seldom studied. More tools are needed for managing other types of TD, such as architectural TD. We think that an ideal TDM tool should be able to handle multiple types of TD, so that a comprehensive management on various types of TD can be performed. Finally, an ideal TDM tool should be integrated into the daily work environment of developers, architects, and project managers.

5.3. Implications for researchers

The results of this mapping study point out a number of implications for researchers.

(1) There should be a boundary between TD and non-TD. Some researchers tend to put anything that is detrimental to the software product and development process under the umbrella of TD, which may cause confusion and ambiguity.
(2) The relationships between TD items should be further investigated. A TD item is not an "island", it can affect or be affected by other TD items. Only S34 and S39 explicitly take the mutual influences between TD items into account.
(3) TDM approaches should be refined significantly, while more empirical studies are needed to show evidence of usage of the different TDM approaches. Many TDM approaches are only mentioned in the related primary studies, but the authors have not deeply discussed or investigated how to use those approaches in actual cases. Thus, these kinds of approaches may not be practical enough to be used in real projects.
(4) TD representation/documentation needs more investigation, as it is currently under-investigated. Some types of TD (e.g., automatic static analysis issues and lack of unit tests) can be automatically identified through code analysis tools, therefore, they can be effectively documented at least in some form. However, some other types of TD (e.g., architectural TD and infrastructure TD) that usually cannot be automatically identified, some effort needs to be spent for their documentation and further management. Identifying such types of TD requires much effort and time, but if not documented, such TD may be overlooked.
(5) Since code-related TD (e.g., code TD and design TD) may change frequently because of fast evolving source code, it is desirable that code-related TD is automatically documented or marked along with the code through e.g., automatic code analysis tools, so that code-related TD can be easily monitored and tracked.
(6) Since TD will evolve when the software system changes over time, we believe that it is necessary to emphasize the traceability between TD and related artifacts, such as architecture and detailed design models, source code, and requirements. For instance, it is useful to document the relationships between architectural TD and related components when codifying architectural TD, for the purpose of estimating the cost of repaying architectural TD.
(7) Most collected TDM tools take source code as input, but source code cannot cover all types of TD, such as infrastructure TD. We think it is fruitful to design and develop tools dealing with artifacts other than source code (e.g., architectural design models and requirements) to manage TD.
(8) There are only seven tools that can be used for other types of TD except for code and design TD. We encourage researchers to consider existing tools (e.g., modeling tools) in other areas of software development and to develop new tools that support managing various types of TD (e.g., architectural TD).
(9) More research is needed to investigate the capabilities of existing IDEs in managing TD and to extend IDEs and design tools (e.g., modeling tools) for managing TD. TDM tools should be integrated into the work environment of development teams, making TDM as part of their daily work instead of an additional task on them.
(10) The results of quality assessment of the selected studies (as shown in Appendix C) reveal that the average evidence level of the claims related to TD is only 0.40 (the mean score of Q1), a relatively low score. This means that we lack empirical studies with a high evidence level. More high-level empirical evidence on TDM will make TD stakeholders (e.g., architects, developers, and managers) more confident in managing TD by applying various TDM approaches and tools. More importantly, it can help to raise the awareness of managing TD by reporting cases of successes or failures of TDM.

5.4. Implications for practitioners

The results of this mapping study also point out a number of implications for practitioners.

(1) There are only few cases that present the whole TDM process in industry, so more industrial cases are needed to show how to deal with different types of TD in practice. These industrial cases may increase awareness of TD and even inspire different stakeholders for a more serious consideration of TDM.
(2) More industrial studies are needed to show how to prioritize a list of TD items to maximize the benefit of a software project and which factors should be considered during TD prioritization in the context of commercial software development.
(3) Many tools can already be used for TD identification based on source code and adopted with a low barrier in the daily work of software engineers. Thus, practitioners can already make use of such tools to detect TD for further management.
(4) Some tools can be used for TD measurement as suggested in the related primary studies. However, most of the tools do not calculate the monetary value of the cost to repay the identified TD. TD is about the future cost of software development, but the accuracy of the calculation of TD is not very high. We believe that the metrics used in these tools can only be used as indicators for some types of TD (e.g., architectural TD). In TD measurement, the mutual influences between TD items are not taken into account, thus, this is another threat to the accuracy of TD calculation. Practitioners should be cautious when making development decisions and release planning based on the TD measurements.
(5) Since code-related TD (e.g., code TD and design TD) tends to change frequently due to code evolution, TDM tools that manage code-related TD should be integrated into IDEs, so that the TDM tools can facilitate the daily work of stakeholders in software development. Hence, it is important that practitioners use existing or develop dedicated TDM tools which can be integrated into their work environment.
(6) More exploration is needed on the abilities of existing IDEs in TDM. Some powerful IDEs, such as Microsoft Visual Studio (MS VS), are helpful in TDM in terms of TD identification, measurement, repayment, etc. For instance, MS VS can facilitate the management of architectural TD, design TD, code TD, and test TD. Practitioners can extend MS VS (e.g., by developing plugins) to combine all the available functionalities supporting various TD types and TDM activities to systematically facilitate an integrated TDM process.

6. Threats to validity

The results of this systematic mapping study may be affected by the coverage of the study search, bias on study selection, imbalance of study distribution, inaccuracy of data extraction, and bias on data synthesis, which are discussed in this section.

6.1. Incompleteness of study search

There may be relevant studies that were not retrieved, which may affect the completeness of the study search. To mitigate this threat, first, we searched the most popular electronic databases in which a large number of journals, conference and workshop proceedings, and book chapters in the software engineering field are indexed. The trial search was performed on nine databases and the final full search was performed on seven databases. Second, we employed the "snowballing" technique (Wohlin, 2014) to include the potential studies in the references of the selected studies retrieved from the database searches. Third, the extension in Google Scholar also helped to increase the completeness of the potentially relevant studies, since Google Scholar may include relevant studies that are not indexed in the selected databases. Although we took actions to improve the completeness of study search, there were still limitations: since we used the word "debt" as the search term, it is possible that we missed relevant studies that investigate the phenomenon of TD but do not explicitly use the term "debt" (e.g., work related to 'smells' or 'refactoring').

6.2. Bias on study selection

There may be bias on behalf of the researchers regarding the study selection. A first step toward reducing the bias is to set clear inclusion and exclusion criteria for study selection. However, different researchers are prone to have different understandings on these criteria, hence the selection results of different researchers tend to be varied. To mitigate the bias on study selection results, a pilot selection was performed to ensure that the researchers reached a consensus on the understanding of the selection criteria. Also, the study protocol was discussed among the researchers to ensure a common understanding on study selection. Moreover, in the second and final round of study selections, two researchers conducted the selection process in parallel and independently, and then compared and discussed their selection results to mitigate any potential personal bias in study selection. However, since we included only peer-reviewed studies in this mapping study, it is possible that we missed some important non-peer-reviewed work on TD (e.g., McConnell's white paper on TD (McConnell, 2008)).

6.3. Imbalance of study distribution over publication venues

As we can see in Appendix B (Distribution of selected studies over publication sources), around one third of the selected studies (31 out of 94) come from the series of the MTD workshops. To some extent, these studies may carry the biases of the workshop organizers and committee members. However, we did not deal with such kind of biases, because there is no effective way to identify exactly what such biases are and thus they cannot be mitigated or eliminated. In addition, workshops by definition allow immature results to be published, which may skew the evidence level of the selected primary studies.

6.4. Inaccuracy of data extraction

Bias on data extraction may result in inaccuracy of the extracted data items, which may affect the classification and analysis results of the selected studies. This bias was reduced by three measures. First, the data items to be extracted in this mapping study were discussed among the researchers and agreement on the meaning of each data item was achieved. For instance, in the study protocol, we explicitly defined what TD-related notions and TDM activities are, and disagreements were resolved among the researchers. Second, a pilot data extraction was performed among three researchers, and disagreements on the results of the pilot data extraction were discussed to reach a consensus. This measure further improved the accuracy of the extracted data items. Third, the data extraction results were checked by two researchers, and again disagreements were discussed and resolved.

6.5. Bias on data synthesis

Not all papers sufficiently and clearly describe the details of information that is to be extracted as data items. Therefore, we had to infer certain pieces of information of data items during data synthesis. For instance, a study may mention that specific QAs are affected when TD is incurred, without providing the definitions or descriptions of these

QAs. Since there is no consensus in the literature on the complete list of QAs and their definitions, we chose to map the collected QAs to the QAs defined in ISO/IEC 25010 [2011]. During this mapping, we needed to infer the real meanings of the QAs which definitions are not specified. To minimize the possibility of the inaccuracy of such inferences, we discussed the extracted data items after each researcher checked their context in the studies from which they were extracted, to clarify the potential ambiguities.

7. Conclusions

In this mapping study, we searched for relevant studies in seven main publication databases and 75 studies got selected. Ten more studies were selected through the snowballing technique. Finally, nine more studies were selected by extension in Google Scholar. In total, we got 94 primary studies finally selected for data extraction. Based on the extracted data, we get a comprehensive understanding of the concept of TD, as well as an overview of the current state of the research on TDM. The main conclusions we draw are summarized in the following points:

(1) Both academia and industry paid significant attention to research on TD, according to the distribution of the selected studies over author types.
(2) Widespread attention was paid to research on TD throughout the software development lifecycle, considering the wide range of publication sources in which the selected studies were published.
(3) The number of the published studies on TD had been increasing significantly from 2008 to 2013.
(4) *Interest*, *principal*, and *risk* are the most frequently-used notions to describe and explain the concept of TD.
(5) TD can be classified into 10 types and each TD type can be further categorized into sub-types according to the causes of TD. The 10 types of TD are requirements TD, architectural TD, design TD, code TD, test TD, build TD, documentation TD, infrastructure TD, versioning TD, and defect TD. Among the 10 TD types, code TD was the most studied in the selected studies.
(6) Most studies argue that TD negatively affects the maintainability (maintainability as a whole or its sub-QAs) of software systems, while other QAs and sub-QAs are only mentioned in a handful of studies each.
(7) The various TDM activities received significantly different levels of attention, with TD repayment, identification, and measurement receiving the most attention and TD representation/documentation the least attention.
(8) The numbers of studies on approaches for different TDM activities vary significantly. The approaches for TD identification, measurement, and repayment were mentioned, used, or proposed the most frequently in the selected studies, while approaches for TD representation/documentation received the least attention. The most discussed approaches for TD identification, measurement, and repayment are code analysis, calculation models, and refactoring, respectively.
(9) Twenty-nine tools for managing TD were collected from the selected studies. Among the 29 tools, only four are dedicated tools to managing TD and the rest are borrowed from other fields of software development. Each of the four dedicated TDM tools can support more than two TDM activities, while the rest of the tools can only support one or two TDM activities. Most tools support code and design TD management, while few tools support managing other types of TD (e.g., architectural TD). Most tools use source code as input.

With the implications discussed in Section 5, we encourage the researchers and practitioners in software engineering community to conduct more empirical studies with high-level evidence on the whole TDM process and on the application of specific TDM approaches in industrial settings. In addition, more sophisticated and dedicated TDM tools are needed for managing various types of TD in the whole TDM process.

Acknowledgments

This work is partially supported by AFR-Luxembourg under the contract no. 895528 and the NSFC under the grant no. 61472286. We are thankful to Apostolos Ampatzoglou and Sara Mahdavi-Hezavehi for reviewing an earlier version of this paper. We are also thankful to the anonymous reviewers for their valuable and constructive comments.

Appendix A. Selected studies

[S1] E. Allman, Managing technical debt – shortcuts that save money and time today can cost you down the road, Communications of the ACM 55 (5) (2012) 50–55.
[S2] E. Alzaghoul, R. Bahsoon, CloudMTD: using real options to manage technical debt in cloud-based service selection, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA 2013, pp. 55–62.
[S3] B. Barton, C. Sterling, Manage project portfolios more effectively by including software debt in the decision process, Cutter IT Journal 23 (10) (2010) 19–24.
[S4] R. Bavani, Distributed agile, agile testing, and technical debt, IEEE Software 29 (6) (2012) 28–33.
[S5] S. Bellomo, R.L. Nord, I. Ozkaya, A study of enabling factors for rapid fielding: combined practices to balance speed and stability, in: Proceedings of the 2013 International Conference on Software Engineering (ICSE'13), IEEE, San Francisco, CA, USA, 2013, pp. 982–991.
[S6] S. Black, P.P. Boca, J.P. Bowen, J. Gorman, M. Hinchey, Formal versus agile: survival of the fittest, Computer 42 (9) (2009) 37–45.
[S7] J. Bohnet, J. Döllner, Monitoring code quality and development activity by software maps, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 9–16.
[S8] J. Brondum, L. Zhu, Visualising architectural dependencies, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 7–14.
[S9] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, N. Zazworka, Managing technical debt in software-reliant systems, in: Proceedings of the FSE/SDP workshop on future of software engineering research (FoSER'10), ACM, Santa Fe, New Mexico, USA, 2010, pp. 47–52.
[S10] F. Buschmann, To pay or not to pay technical debt, IEEE Software 28 (6) (2011) 29–31.
[S11] S. Chin, E. Huddleston, W. Bodwell, I. Gat, The economics of technical debt, Cutter IT Journal 23 (10) (2010) 11–15.
[S12] Z. Codabux, B. Williams, Managing technical debt: an industrial case study, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 8–15.
[S13] P. Conroy, Technical debt: where are the shareholders' interests?, IEEE Software 29 (6) (2012) 88–88.
[S14] T. Coq, J.P. Rosen, The SQALE quality and analysis models for assessing the quality of Ada source code, in: Proceedings of the 16th Ada-Europe International Conference on Reliable Software Technologies (Ada-Europe'11), Springer Berlin Heidelberg, Edinburgh, UK, 2011, pp. 61–74.

[S15] W. Cunningham, The WyCash portfolio management system, in: Proceedings of the 7th Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92), ACM, Vancouver, British Columbia, Canada, 1992, pp. 29–30.
[S16] B. Curtis, J. Sappidi, A. Szynkarski, Estimating the principal of an application's technical debt, IEEE Software 29 (6) (2012) 34–42.
[S17] B. Curtis, J. Sappidi, A. Szynkarski, Estimating the size, cost, and types of Technical Debt, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 49–53.
[S18] J.D. Davis, T.J. Andersen, Surviving the economic downturn, in: Proceedings of the 12th AGILE Conference (AGILE'09), IEEE, Chicago, IL, USA, 2009, pp. 245–250.
[S19] N. Davis, Driving quality improvement and reducing technical debt with the definition of done, in: Proceedings of the 16th AGILE Conference (AGILE'13), IEEE, Nashville, TN, USA, 2013, pp. 164–168.
[S20] J. de Groot, A. Nugroho, T. Back, J. Visser, What is the value of your software?, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 37–44.
[S21] C. Ebert, A useful metaphor for risk – poorly practiced, IEEE Software 29 (6) (2012) 53–55.
[S22] R.J. Eisenberg, A threshold based approach to technical debt, SIGSOFT Software Engineering Notes 37 (2) (2012) 1–6.
[S23] N.A. Ernst, On the role of requirements in understanding and managing technical debt, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 61–64.
[S24] D. Falessi, M.A. Shaw, F. Shull, K. Mullen, M.S. Keymind, Practical considerations, challenges, and requirements of tool-support for managing technical debt, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 16–19.
[S25] F.A. Fontana, V. Ferme, S. Spinelli, Investigating the impact of code smells debt on quality code evaluation, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 15–22.
[S26] I. Gat, Revolution in software: using technical debt techniques to govern the software development process, Agile Product & Project Management 11 (4) (2010) 1–13.
[S27] I. Gat, Technical debt assessment: a case of simultaneous improvement at three levels, Agile Product & Project Management, 11 (10) (2010) 1–4.
[S28] I. Gat, Technical debt as a meaningful metaphor for code quality, IEEE Software 29 (6) (2012) 52–54.
[S29] I. Gat, J.D. Heintz, From assessment to reduction: how cutter consortium helps rein in millions of dollars in technical debt, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 24–26.
[S30] J.M. Golden, Transformation patterns for curing the human causes of technical debt, Cutter IT Journal 23 (10) (2010) 30–35.
[S31] R. Gomes, C. Siebra, G. Tonin, A. Cavalcanti, F.Q.B.d. Silva, A.L.M. Santos, R. Marques, An extraction method to collect data on defects and effort evolution in a constantly modified system, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 27–30.
[S32] D.R. Greening, Release duration and enterprise agility, in: Proceedings of the 46th Hawaii International Conference on System Sciences (HICSS'13), IEEE, Wailea, HI, USA, 2013, pp. 4835–4841.
[S33] I. Griffith, C. Izurieta, Design pattern decay: an extended taxonomy and empirical study of grime and its impact on design pattern evolution, in: Proceedings of Empirical Software Engineering International Week, ACM, Baltimore, Maryland, USA, 2013.
[S34] Y. Guo, C. Seaman, A portfolio approach to technical debt management, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 31–34.
[S35] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F.Q.B. Da Silva, A.L.M. Santos, C. Siebra, Tracking technical debt – an exploratory case study, in: Proceedings of the 27th IEEE International Conference on Software Maintenance (ICSM'11), IEEE, Williamsburg, VI, USA, 2011, pp. 528–531.
[S36] Y. Guo, C. Seaman, N. Zazworka, F. Shull, Domain-specific tailoring of code smells: an empirical study, in: Proceedings of the 32nd International Conference on Software Engineering (ICSE'10), ACM, Cape Town, South Africa, 2010, pp. 167–170.
[S37] J. Heidenberg, I. Porres, Metrics functions for Kanban guards, in: Proceedings of the 17th IEEE International Conference and Workshops on Engineering of Computer Based Systems (ECBS'10), IEEE, Oxford, England, UK, 2010, pp. 306–310.
[S38] J. Heintz, Modernizing the delorean system:comparing actual and predicted results of a technical debt reduction project, Cutter IT Journal 23 (10) (2010) 7–10.
[S39] J. Holvitie, V. Leppanen, DebtFlag: technical debt management with a development environment integrated tool, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 20–27.
[S40] C. Izurieta, I. Griffith, D. Reimanis, R. Luhr, On the uncertainty of technical debt measurements, in: Proceedings of the 4th International Conference on Information Science and Applications (ICISA'13), IEEE, Suwon, South Korea, 2013, pp. 1–4.
[S41] C. Izurieta, A. Vetrò, N. Zazworka, Y. Cai, C. Seaman, F. Shull, Organizing the technical debt landscape, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 23–26.
[S42] M. Kaiser, G. Royse, Selling the investment to pay down technical debt: the code Christmas tree, in: Proceedings of the 14th AGILE Conference (AGILE'11), IEEE, Salt Lake City, UT, USA, 2011, pp. 175–180.
[S43] M. Karlesky, W. Bereza, G. Williams, M. Fletcher, Mocking the embedded world: test-driven development, continuous integration, and design patterns, in: Proceedings of Embedded Systems Conference Silicon Valley, San Jose, CA, USA, 2007.
[S44] T. Klinger, P. Tarr, P. Wagstrom, C. Williams, An enterprise perspective on technical debt, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 35–38.
[S45] S. Koolmanojwong, J.A. Lane, Enablers and inhibitors of expediting systems engineering, Procedia Computer Science 16 (2013) 483–491.
[S46] V. Krishna, A. Basu, Minimizing technical debt: developer's viewpoint, in: Proceedings of the International Conference on Software Engineering and Mobile Application Modelling and Development (ICSEMA'12), IET, Chennai, India, 2012, pp. 1–5.
[S47] P. Kruchten, R.L. Nord, I. Ozkaya, Technical debt: from metaphor to theory and practice, IEEE Software 29 (6) (2012) 18–21.
[S48] P. Kruchten, R.L. Nord, I. Ozkaya, D. Falessi, Technical debt: towards a crisper definition report on the 4th International Workshop on Managing Technical Debt, SIGSOFT Software Engineering Notes 38 (5) (2013) 51–54.
[S49] O. Ktata, G. Lévesque, Designing and implementing a measurement program for Scrum teams: what do agile developers really need and want?, in: Proceedings of the 3rd C* Conference

on Computer Science and Software Engineering (C3S2E'10), ACM, Montreal, Quebec, Canada, 2010, pp. 101–107.
[S50] J.-L. Letouzey, The SQALE method for evaluating technical debt, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 31–36.
[S51] J.-L. Letouzey, M. Ilkiewicz, Managing technical debt with the SQALE method, IEEE Software 29 (6) (2012) 44–51.
[S52] E. Lim, N. Taksande, C. Seaman, A balancing act: what software practitioners have to say about technical debt, IEEE Software 29 (6) (2012) 22–27.
[S53] M. Lindgren, R. Land, C. Norstrom, A. Wall, Key aspects of software release planning in industry, in: Proceedings of the 19th Australian Conference on Software Engineering (ASWEC'08), IEEE, Perth, Australia, 2008, pp. 320–329.
[S54] M. Lindgren, A. Wall, R. Land, C. Norstrom, A method for balancing short- and long-term investments: quality vs. features, in: Proceedings of the 34th Euromicro Conference Software Engineering and Advanced Applications (SEAA'08), IEEE, Parma, Italy, 2008, pp. 175–182.
[S55] R. Marinescu, Assessing technical debt by identifying design flaws in software systems, IBM Journal of Research and Development 56 (5) (2012) 9:1–9:13.
[S56] J.D. McGregor, J.Y. Monteith, Z. Jie, Technical debt aggregation in ecosystems, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 27–30.
[S57] R. Mo, J. Garcia, Y. Cai, N. Medvidovic, Mapping architectural decay instances to dependency models, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 39–46.
[S58] J.Y. Monteith, J.D. McGregor, Exploring software supply chains from a technical debt perspective, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 32–38.
[S59] J.D. Morgenthaler, M. Gridnev, R. Sauciuc, S. Bhansali, Searching for build debt: experiences managing technical debt at Google, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 1–6.
[S60] C.J. Neill, P.A. Laplante, Paying down design debt with strategic refactoring, Computer 39 (12) (2006) 131–134.
[S61] R.L. Nord, I. Ozkaya, P. Kruchten, M. Gonzalez-Rojas, In search of a metric for managing architectural technical debt, in: Proceedings of the 10th Working IEEE/IFIP Conference on Software Architecture (WICSA'12), IEEE, Helsinki, Finland, 2012.
[S62] A. Nugroho, J. Visser, T. Kuipers, An empirical model of technical debt and interest, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 1–8.
[S63] D. O'Connor, Technical debt in semiconductor equipment: it's time to pay it down, Solid State Technology 53 (7) (2010) 34–35.
[S64] K. Power, Understanding the impact of technical debt on the capacity and velocity of teams and organizations: viewing team and organization capacity as a portfolio of real options, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 28–31.
[S65] K. Pugh, The risks of acceptance test debt, Cutter IT Journal 23 (10) (2010) 25–29.
[S66] N. Ramasubbu, C.F. Kemerer, Towards a model for optimizing technical debt in software products, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 51–54.
[S67] D. Rooney, Technical debt: challenging the metaphor, Cutter IT Journal 23 (10) (2010) 16–18.
[S68] Y. Rubin, S. Kallner, N. Guy, G. Shachor, Restraining technical debt when developing large-scale Ajax applications, in: The First International Conference on Building and Exploring Web Based Environments (WEB'13) 2013, pp. 13–18.
[S69] P. Santos, A. Varella, C. Dantas, D. Borges, Visualizing and managing technical debt in agile development: an experience report, in: H. Baumeister, B. Weber (Eds.), Proceedings of the 14th International Conference on Agile Software Development (XP'13), Springer Berlin Heidelberg, Vienna, Austra, 2013, pp. 121–134.
[S70] K. Schmid, A formal approach to technical debt decision making, in: Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures (QoSA'13), ACM, Vancouver, British Columbia, Canada, 2013, pp. 153–162.
[S71] K. Schmid, On the limits of the technical debt metaphor some guidance on going beyond, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 63–66.
[S72] C. Seaman, Y. Guo, Measuring and monitoring technical debt, in: M. Zelkowitz (Ed.), Advances in Computers, Elsevier, 2011, pp. 25–45.
[S73] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, A. Vetrò, Using technical debt data in decision making: potential decision approaches, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 45–48.
[S74] A.C. Shafer, Infrastructure debt: revisiting the foundation, Cutter IT Journal 23 (10) (2010) 36–41.
[S75] S. Shah, M. Torchiano, A. Vetro, M. Morisio, Exploratory testing as a source of testing technical debt, IT Professional PP (99) (2013) 1–1.
[S76] F. Shull, Perfectionists in a world of finite resources, IEEE Software 28 (2) (2011) 4–6.
[S77] F. Shull, D. Falessi, C. Seaman, M. Diep, L. Layman, Technical debt: showing the way for better transfer of empirical results, in: J. Münch, K. Schmid (Eds.), Perspectives on the Future of Software Engineering, Springer Berlin Heidelberg, Berlin, Germany, 2013, pp. 179–190.
[S78] C.A. Siebra, G.S. Tonin, F.Q.B. da Silva, R.G. Oliveira, L.C. Antonio, R.C.G. Miranda, A.L.M. Santos, Managing technical debt in practice: an industrial report, in: Proceedings of the 6th ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM'12), ACM, Lund, Scania, Sweden, 2012, pp. 247–250.
[S79] R. Smith, Computer system design, Journal of GXP Compliance 13 (2009) 44–48.
[S80] W. Snipes, B. Robinson, Y. Guo, C. Seaman, Defining the decision factors for managing defects: a technical debt perspective, in: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12), IEEE, Zurich, Switzerland, 2012, pp. 54–60.
[S81] R.O. Spínola, N. Zazworka, A. Vetrò, C. Seaman, F. Shull, Investigating technical debt folklore: shedding some light on technical debt opinion, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 1–7.
[S82] C. Sterling, Technical debt, in: Managing Software Debt – Building for Inevitable Change, Addison-Wesley Professional, Boston, USA, 2010, pp. 15–30.
[S83] M.G. Stochel, M.R. Wawrowski, M. Rabiej, Value-based technical debt model and its application, in: Proceedings of the 7th International Conference on Software Engineering Advances

(ICSEA'12), Xpert Publishing Services, Lisbon, Portugal, 2012, pp. 205–212.
[S84] S. Stolberg, Enabling agile testing through continuous integration, in: Proceedings of the 12th Agile Conference (AGILE'09), IEEE, Hannover, Germany, 2009, pp. 369–374.
[S85] T. Theodoropoulos, M. Hofberg, D. Kern, Technical debt from the stakeholder perspective, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 43–46.
[S86] C. Verhoef, How to implement the future?, in: Proceedings of the 26th Euromicro Conference (EURMIC'00), vol. 31, IEEE, Maastricht, The Netherlands, 2000, pp. 32–47.
[S87] A. Vetro, Using automatic static analysis to identify technical debt, in: Proceedings of the 34th International Conference on Software Engineering (ICSE'12), IEEE, Zurich, Switzerland, 2012, pp. 1613–1615.
[S88] P. Wang, J. Yang, L. Tan, R. Kroeger, J.D. Morgenthaler, Generating precise dependencies for large software, in: Proceedings of the 4th International Workshop on Managing Technical Debt (MTD'13), IEEE, San Francisco, CA, USA, 2013, pp. 47–50.
[S89] K. Wiklund, S. Eldh, D. Sundmark, K. Lundqvist, Technical debt in test automation, in: Proceedings of the 5th IEEE International Conference on Software Testing, Verification and Validation (ICST'12), IEEE, Montreal, QC, Canada, 2012, pp. 887–892.
[S90] J. Xuan, Y. Hu, H. Jiang, Debt-prone bugs: technical debt in software maintenance, International Journal of Advancements in Computing Technology 4 (19) (2012) 453–461.
[S91] N. Zazworka, C. Seaman, F. Shull, Prioritizing design debt investment opportunities, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 39–42.
[S92] N. Zazworka, M.A. Shaw, F. Shull, C. Seaman, Investigating the impact of design debt on software quality, in: Proceedings of the 2nd International Workshop on Managing Technical Debt (MTD'11), ACM, Waikiki, Honolulu, HI, USA, 2011, pp. 17–23.
[S93] N. Zazworka, R.O. Spinola, A. Vetro', F. Shull, C. Seaman, A case study on effectively identifying technical debt, in: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE'13), ACM, Porto de Galinhas, Brazil, 2013, pp. 42–47.
[S94] N. Zazworka, A. Vetro', C. Izurieta, S. Wong, Y. Cai, C. Seaman, F. Shull, Comparing four approaches for technical debt identification, Software Quality Journal (2013) 1–24.

Appendix B. Distribution of selected studies over publication sources

#	Publication source	Type	No.	%
1	Managing Technical Debt Workshop	Workshop	31	33
2	IEEE Software	Journal	10	11
3	Cutter IT Journal	Journal	7	7
4	Agile Conference	Conference	4	4
5	International Conference on Software Engineering	Conference	3	3
6	Computer	Journal	2	2
7	SIGSOFT Software Engineering Notes	Journal	2	2
8	Agile Product & Project Management	Journal	2	2
9	International Symposium on Empirical Software Engineering and Measurement	Conference	1	1
10	Ada-Europe	Conference	1	1
11	Advances in Computers	Book chapter	1	1
12	Australian Software Engineering Conference	Conference	1	1
13	C* Conference on Computer Science & Software Engineering	Conference	1	1
14	Communications of the ACM	Journal	1	1
15	Embedded Systems Conference Silicon Valley	Conference	1	1
16	Euromicro Conference	Conference	1	1
17	Euromicro Conference on Software Engineering and Advanced Applications	Conference	1	1
18	FSE/SDP workshop on Future of Software Engineering Research	Workshop	1	1
19	Hawaii International Conference on System Sciences	Conference	1	1
20	IBM Journal of Research and Development	Journal	1	1
21	IEEE International Conference and Workshops on Engineering of Computer Based Systems	Conference	1	1
22	International Conference on Software Maintenance	Conference	1	1
23	International Conference on Software Testing, Verification and Validation	Conference	1	1
24	International ACM Sigsoft Conference on the Quality of Software Architectures	Conference	1	1
25	Agile Processes in Software Engineering and Extreme Programming	Conference	1	1
26	International Conference on Building and Exploring Web Based Environments	Conference	1	1
27	International Conference on Evaluation and Assessment in Software Engineering	Conference	1	1
28	International Conference on Information Science and Applications	Conference	1	1
29	International Conference on Software Engineering Advances	Conference	1	1
30	International Conference on Software Engineering and Mobile Application Modelling and Development	Conference	1	1
31	International Doctoral Symposium on Empirical Software Engineering	Conference	1	1
32	International Journal of Advancements in Computing Technology	Journal	1	1
33	IT Professional	Journal	1	1
34	Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture	Conference	1	1
35	Journal of GXP Compliance	Journal	1	1
36	Managing Software Debt – Building for Inevitable Change	Book chapter	1	1
37	Object oriented programming systems, languages, and applications	Conference	1	1
38	Perspectives on the Future of Software Engineering	Book chapter	1	1
39	Procedia Computer Science	Journal	1	1
40	Software Quality Journal	Journal	1	1
41	Solid State Technology	Journal	1	1
	Total		94	100

Appendix C. Quality assessment results of the selected primary studies

Study	Q1	Q2	Q3	Q4	Q5	Sum	Study	Q1	Q2	Q3	Q4	Q5	Sum
S1	0.00	1.00	1.00	1.00	0.00	3.00	S48	0.40	1.00	1.00	1.00	0.00	3.40
S2	0.20	1.00	1.00	1.00	0.00	3.20	S49	0.00	1.00	1.00	1.00	0.00	3.00
S3	0.00	1.00	1.00	1.00	0.00	3.00	S50	0.00	1.00	1.00	1.00	0.50	3.50
S4	0.40	1.00	0.00	1.00	0.00	2.40	S51	0.20	1.00	1.00	1.00	0.00	3.20
S5	0.80	1.00	1.00	0.50	0.00	3.30	S52	0.80	1.00	1.00	0.00	1.00	3.80
S6	0.40	1.00	0.00	0.00	0.00	1.40	S53	0.80	1.00	1.00	0.00	0.00	2.80
S7	0.80	1.00	0.00	1.00	0.00	2.80	S54	0.80	1.00	0.00	0.00	0.00	1.80
S8	0.60	1.00	1.00	1.00	0.00	3.60	S55	0.80	1.00	1.00	1.00	1.00	4.80
S9	0.40	1.00	1.00	1.00	0.00	3.40	S56	0.00	1.00	1.00	1.00	0.00	3.00
S10	0.00	1.00	1.00	0.00	0.00	2.00	S57	0.20	1.00	1.00	1.00	0.00	3.20
S11	0.00	1.00	1.00	0.00	0.00	2.00	S58	0.60	1.00	0.50	1.00	0.00	3.10
S12	0.80	1.00	1.00	1.00	1.00	4.80	S59	1.00	1.00	1.00	1.00	0.00	4.00
S13	0.00	1.00	0.00	0.00	0.00	1.00	S60	0.60	1.00	1.00	1.00	0.00	3.60
S14	0.20	1.00	0.00	1.00	0.00	2.20	S61	0.20	1.00	1.00	1.00	1.00	4.20
S15	1.00	1.00	1.00	1.00	0.00	4.00	S62	0.80	1.00	1.00	1.00	1.00	4.80
S16	0.80	1.00	1.00	1.00	0.00	3.80	S63	0.00	1.00	0.00	1.00	0.00	2.00
S17	0.80	1.00	1.00	1.00	0.00	3.80	S64	0.20	1.00	0.50	0.00	0.00	1.70
S18	1.00	1.00	0.00	0.50	0.00	2.50	S65	0.00	1.00	1.00	1.00	0.00	3.00
S19	1.00	1.00	0.50	1.00	0.50	4.00	S66	0.00	1.00	1.00	0.50	0.00	2.50
S20	0.80	1.00	1.00	0.00	1.00	3.80	S67	0.00	1.00	1.00	1.00	0.00	3.00
S21	0.00	1.00	1.00	0.00	0.00	2.00	S68	0.60	1.00	1.00	1.00	0.00	3.60
S22	0.20	1.00	1.00	1.00	0.50	3.70	S69	0.80	1.00	1.00	1.00	0.00	3.80
S23	0.00	1.00	1.00	1.00	1.00	4.00	S70	0.20	1.00	0.50	1.00	0.00	2.70
S24	0.40	1.00	1.00	1.00	0.00	3.40	S71	0.00	1.00	0.50	0.50	0.00	2.00
S25	0.60	1.00	0.00	1.00	0.00	2.60	S72	0.20	1.00	1.00	1.00	0.00	3.20
S26	0.20	1.00	0.50	1.00	0.00	2.70	S73	0.00	1.00	1.00	0.00	0.00	2.00
S27	0.80	1.00	0.50	1.00	0.00	3.30	S74	0.00	1.00	1.00	1.00	0.00	3.00
S28	0.00	1.00	0.00	0.00	0.00	1.00	S75	0.00	1.00	1.00	1.00	0.00	3.00
S29	0.80	1.00	0.00	1.00	0.00	2.80	S76	0.00	1.00	1.00	1.00	0.00	3.00
S30	1.00	1.00	0.00	0.00	0.00	2.00	S77	0.00	1.00	1.00	1.00	0.00	3.00
S31	0.80	1.00	0.50	1.00	0.00	3.30	S78	1.00	1.00	1.00	1.00	0.00	4.00
S32	0.00	1.00	1.00	1.00	0.00	3.00	S79	0.20	1.00	0.00	0.00	0.00	1.20
S33	0.00	1.00	1.00	1.00	1.00	4.00	S80	0.80	1.00	0.00	1.00	1.00	3.80
S34	0.00	1.00	1.00	1.00	0.50	3.50	S81	0.60	1.00	1.00	0.00	1.00	3.60
S35	0.80	1.00	1.00	1.00	1.00	4.80	S82	0.00	1.00	1.00	1.00	0.00	3.00
S36	0.80	1.00	1.00	1.00	1.00	4.80	S83	0.20	1.00	1.00	1.00	0.00	3.20
S37	0.00	1.00	1.00	1.00	0.00	3.00	S84	1.00	1.00	1.00	1.00	0.00	4.00
S38	0.80	1.00	0.50	1.00	0.00	3.30	S85	0.00	1.00	1.00	1.00	0.00	3.00
S39	0.00	1.00	0.50	1.00	1.00	3.50	S86	0.00	1.00	0.00	1.00	0.00	2.00
S40	0.00	1.00	1.00	0.00	0.00	2.00	S87	0.60	1.00	1.00	1.00	0.00	3.60
S41	0.00	1.00	0.00	1.00	0.00	2.00	S88	0.60	1.00	0.00	1.00	0.00	2.60
S42	0.80	1.00	1.00	1.00	0.50	4.30	S89	0.80	1.00	1.00	1.00	0.00	3.80
S43	0.20	1.00	1.00	1.00	0.00	3.20	S90	0.60	1.00	1.00	1.00	1.00	4.60
S44	0.40	1.00	1.00	1.00	0.00	3.40	S91	0.20	1.00	1.00	1.00	0.00	3.20
S45	0.00	1.00	1.00	0.00	0.00	2.00	S92	0.80	1.00	1.00	1.00	1.00	4.80
S46	1.00	1.00	1.00	1.00	0.00	4.00	S93	0.80	1.00	1.00	1.00	1.00	4.80
S47	0.40	1.00	1.00	0.00	0.00	2.40	S94	0.60	1.00	1.00	1.00	1.00	4.60
		Q1		Q2		Q3		Q4		Q5		Sum	
Mean score		0.40		1.00		0.77		0.79		0.21		3.17	

The study quality assessment questions are as follows:
Q1: How much evidence supports the claims related to TD in the study?
Q2: Is there a clear statement of the aims of the research?
Q3: Is there a clear statement of the definition of TD?
Q4: Is there a clear statement of which types of TD the paper focuses on?
Q5: Are the limitations of the study discussed explicitly?

References

Ali, M.S., Ali Babar, M., Chen, L., Stol, K.-J., 2010. A systematic review of comparative evidence of aspect-oriented programming. Inform. Softw. Technol. 52 (9), 871–887.

Allman, E., 2012. Managing technical debt—shortcuts that save money and time today can cost you down the road. Commun. ACM 55 (5), 50–55.

Alves, V., Niu, N., Alves, C., Valença, G, 2010. Requirements engineering for software product lines: a systematic literature review. Inform. Softw. Technol. 52 (8), 806–820.

Barton, B., Sterling, C., 2010. Manage project portfolios more effectively by including software debt in the decision process. Cutter IT J. 23 (10), 19–24.

Basili, V.R., 1992. Software Modeling and Measurement: The Goal/Question/Metric Paradigm. University of Maryland at College Park, 24.

Birkeland, J., 2010. From a Timebox tangle to a more flexible flow. In: Sillitti, A., Martin, A., Wang, X., Whitworth, E. (Eds.), Proceedings of the 12th International Conference on Agile Software Development (XP '11). Springer, Berlin, Heidelberg, MadridSpain, pp. 325–334.

Brown, N., Cai, Y., Guo, Y., Kazman, R., Kim, M., Kruchten, P., Lim, E., MacCormack, A., Nord, R., Ozkaya, I., Sangwan, R., Seaman, C., Sullivan, K., Zazworka, N., 2010. Managing technical debt in software-reliant systems. In: Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research (FoSER'10). ACM, Santa Fe, New Mexico, USA, pp. 47–52.

Budgen, D., Turner, M., Brereton, P., Kitchenham, B., 2008. Using mapping studies in software engineering. In: Proceedings of PPIG 2008. Lancaster University, UK, pp. 195–204.

Chen, L., Ali Babar, M., Zhang, H., 2010. Towards an evidence-based understanding of electronic data sources. In: Proceedings of the 14th International Conference on Evaluation and Assessment in Software Engineering (EASE'10). ACM, Keele University, UK.

Cunningham, W., 1992. The WyCash portfolio management system. In: Proceedings of the 7th Object-Oriented Programming Systems, Languages, and Applications (OOPSLA'92). ACM, Vancouver, British Columbia, Canada, pp. 29–30.

Dybå, T., Dingsøyr, T., 2008. Empirical studies of agile software development: a systematic review. Inform. Softw. Technol. 50 (9–10), 833–859.

Engström, E., Runeson, P., 2011. Software product line testing – a systematic mapping study. Inform. Softw. Technol. 53 (1), 2–13.
Ernst, N.A., 2012. On the role of requirements in understanding and managing technical debt. In: Proceedings of the 3rd International Workshop on Managing Technical Debt (MTD'12). IEEE, Zurich, Switzerland, pp. 61–64.
Fowler, M., 2009. Technical debt quadrant. URL: http://martinfowler.com/bliki/TechnicalDebtQuadrant.html (accessed 18.10.14).
Fowler, M., Beck, K., Brant, J., Opdyke, W., Roberts, D., 1999. Refactoring: Improving the Design of Existing Code Addison-Wesley Professional. Berkeley, CA, USA.
Galster, M., Weyns, D., Tofan, D., Michalik, B., Avgeriou, P., 2014. Variability in software systems – a systematic literature review. IEEE Trans. Softw. Eng. 40 (3), 282–306.
ISO/IEC, 2001. Software Engineering – Product Quality. Part 1. Quality Model. ISO.
ISO/IEC, 2011. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — system and software quality models, ISO/IEC FDIS 25010:2011, pp. 1–34.
Kitchenham, B., Charters, S., 2007. Guidelines for Performing Systematic Literature Reviews in Software Engineering, Version 2.3. (EBSE Technical Report EBSE-2007-01). Keele University and Durham University
Kruchten, P., Nord, R.L., Ozkaya, I., Falessi, D., 2013. Technical debt: towards a crisper definition report on the 4th international workshop on managing technical debt. SIGSOFT Softw. Eng. Notes 38 (5), 51–54.
Li, Z., Liang, P., Avgeriou, P., 2013. Application of knowledge-based approaches in software architecture: a systematic mapping study. Inform. Softw. Technol. 55 (5), 777–794.
Lim, E., Taksande, N., Seaman, C., 2012. A balancing act: what software practitioners have to say about technical debt. IEEE Softw. 29 (6), 22–27.
Marinescu, R., 2012. Assessing technical debt by identifying design flaws in software systems. IBM J. Res. Dev. 56 (5), 9:1-9:13.
Martin, A., Biddle, R., Noble, J., 2009. XP customer practices: a grounded theory. In: Proceedings of the 12 AGILE Conference (AGILE '09). IEEE, Chicago, ILUSA, pp. 33–40.
McConnell, S., 2008. Managing technical debt. In: Construx, pp. 1–14 http://www.construx.com/uploadedFiles/Construx/Construx_Content/Resources/Documents/Managing%20Technical%20Debt.pdf(accessed 18.10.14).
Petersen, K., Feldt, R., Mujtaba, S., Mattsson, M., 2008. Systematic mapping studies in software engineering. In: Proceedings of the 12th International Conference on Evaluation and Assessment in Software Engineering (EASE'08). ACM, University of Bari, Italy.
Rooney, D., 2010. Technical debt: challenging the metaphor. Cutter IT J. 23 (10), 16–18.
Shahin, M., Liang, P., Babar, M.A., 2014. A systematic review of software architecture visualization techniques. J. Syst. Softw. 94 (8), 161–185.
Sutherland, J., Schoonheim, G., Rijk, M., 2009. Fully distributed scrum: replicating local productivity and quality with offshore teams. In: Proceedings of the 42nd Hawaii International Conference on System Sciences (HICSS '09). IEEE, Big Island, HIUSA, pp. 1–8.
Tom, E., Aurum, A., Vidgen, R., 2012. A consolidated understanding of technical debt. In: Proceedings of the 20th European Conference on Information S ystems (ECIS'12). AIS Electronic Library, Barcelona, Spain.
Tom, E., Aurum, A., Vidgen, R., 2013. An exploration of technical debt. J. Syst. Softw. 86 (6), 1498–1516.
Uy, E., Ioannou, N., 2008. Growing and sustaining an offshore scrum engagement. In: Proceedings of the 11th AGILE Conference (AGILE '08). IEEE, Toronto, ON, Canada, pp. 345–350.
Wohlin, C., 2014. Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering (EASE'14). ACM, London, UK.

Zengyang Li is a PhD candidate in the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. He is working in the research fields of technical debt management and architectural knowledge. Before becoming a PhD student, he had worked as a senior software engineer in the telecommunications industry for more than three years. Before joining in industry, he did his master in software engineering and bachelor in information and computing science from Wuhan University, China. He has published several peer-reviewed articles in international journals, conference and workshop proceedings, and books.

Paris Avgeriou is a professor of software engineering in the Department of Mathematics and Computing Science, University of Groningen, the Netherlands where he has led the software engineering research group since 2006. He has co-organized several international conferences and workshops and sits on the editorial board of Springer TPLOP. He has edited special issues in IEEE Software, Elsevier JSS and Springer TPLOP. He has published more than 120 peer-reviewed articles in international journals, conference proceedings and books. His research interests lie in the area of software architecture, with strong emphasis on architecture modeling, knowledge, evolution and patterns.

Peng Liang is a professor of software engineering in the State Key Lab of Software Engineering (SKLSE), School of Computer, Wuhan University, China. He is a visiting researcher at VU University Amsterdam, the Netherlands. Between 2007 and 2009, he was a post-doctoral researcher at the Software Engineering and Architecture (SEARCH) research group at the University of Groningen, the Netherlands. His research interests concern the area of software architecture and requirements engineering. He has published more than 50 articles in peer-reviewed international journals, conference and workshop proceedings, and books.