An Analysis of Automated Technical Debt Measurement

Ilya Khomyakov, Zufar Makhmutov, Ruzilya Mirgalimova, and Alberto Sillitti(✉)

Innopolis University, Innopolis, Russian Federation
{i.khomyakov,z.makhmutov,r.mirgalimova,a.sillitti}@innopolis.ru

Abstract. *Background*: Measuring and understanding Technical Debt (TD) is quite complex since there are a number of different definitions and techniques that have been proposed in the last few years and it is not clear which ones should be used in which conditions. The approaches proposed are almost never based on the existing ones and their validation is often performed in a very limited number of projects. For this reasons, practitioners are confused and find difficult to apply such approaches in their projects.

Goals: This paper investigates the available techniques for evaluating TD using automated tools aiming at helping practitioners and researcher in understanding the available options and apply them correctly.

Method: The study has been performed as a Systematic Literature Review (SLR) applied to 835 studies obtained from the three largest digital libraries and databases.

Results: After applying all filtering stages, 38 papers out of 835 have been selected and analyzed in depth. Almost all of them propose novel approaches to measure TD using different criteria and they do not extend or validate existing approaches.

Conclusions: The area is not mature and it lacks independent evaluations of the models proposed. Authors focus on proposing new approaches and no consolidation can be identified. Moreover, almost all the approaches proposed are automated only partially and through prototype tools designed just to support the studies analyzed in the paper in which the approach is proposed and rarely maintained. These facts makes difficult the application of such methods by practitioners.

Keywords: Technical debt · Measurement · Literature review · Tools

1 Introduction

TD is one of the most recent concepts that has been introduced in software engineering. It acknowledges the trade-off between code quality and the need to meet market expectations (e.g., low costs, short time-to-market, etc.). This is a typical situation for startup companies that have strict requirements to produce a Minimum Viable Product (MVP) to test the market and get funding

J. Filipe et al. (Eds.): ICEIS 2019, LNBIP 378, pp. 250–273, 2020.
https://doi.org/10.1007/978-3-030-40783-4_12

to survive. In such contexts, the sub-optimal decisions that decrease the quality of the system leading to the creation of strategic TD [72] could be a key strategy to achieve success. In any case, companies should understand that such sub-optimal decisions require additional effort to fix the product in the long run [20,21]. However, creating TD could be a valuable strategy to push products on the market knowing that the debt needs to be payed (with interests) in the future.

This phenomenon was originally described by Ward Cunningham in 1992 [22] introducing the concept of TD. There are many more sources of TD that have been investigated recently that involve communication, collaboration among team members, documentation, and individual attitudes [37,72].

Since TD is a way of measuring the effort needed to achieve top quality in a software system compared to the current status, it is of paramount importance being able of measuring (or estimating) it. The importance of such an activity is proved by the simple fact that most of the software projects have some TD [25]. Being able to estimate TD allows development teams and managers to plan the work properly.

It may also happen that TD is too high to be payed [18], requiring different approaches to address it (e.g., rewriting the system). However, knowing that and how the system reached that condition could help in the identification of mistakes and improve the development process.

Frequent changes of software artifacts (mainly in the source code) without corresponding quality assurance measures quickly leads to a decrease in software quality, with an increase in the costs for further development and evolution due to the increase TD [19]. Moreover, the evaluation of TD should be performed automatically to avoid increasing the load of the developers and being able to monitor that continuously during any phase of the development. This is particularly useful in conjunction with the usage of Agile approaches since their delivery-oriented nature and continuous adaptation to the needs of the customer can be more prone to generate TD compared to traditional software development. However, they are also more prone to pay TD through the a proper implementation of refactoring.

For all these reasons, being able of measuring TD automatically is of paramount importance to support the daily work of developers. There are many different approaches to TD in literature and this paper provides an extensive analysis pointing out the current status of the research extending the work the same authors in [33]. In this paper, we have enhanced the analysis including a wider number of primary studies.

The paper is organized as follows: Sect. 2 describes the adopted methodology; Sect. 3 discusses the findings; Sect. 4 investigates the related work; Sect. 5 analyzes the threats to validity; finally, Sect. 6 draws the conclusions and introduces future work.

2 Methodology

The protocol adopted for this Systematic Literature Review (SLR) is the one introduced by Kitchenham and Charters [34] for performing such reviews in the software engineering area.

The main goal of this work is to review the existing studies and highlight the aspects related to TD measurement, therefore we have defined the following research questions:

- RQ1: Which are the existing techniques for measuring TD?
- RQ2: Which are the tools that support the automation of the measurement of TD?
- RQ3: Are there any empirical studies able to demonstrate the usefulness of the identified techniques?
- RQ4: Are there any empirical studies able to demonstrate the usefulness of the tools identified?

To answer the research questions, we have searched for papers using the three largest digital libraries: ACM Digital Library, IEEE Xplore, and Google Scholar.

Since only studies focusing on TD as main topic are interesting for our purpose, we suppose that their title or abstract include the key words *technical debt measurement*. Consequently, we used appropriate queries for each library:

- ACM Digital Library:(+technical +debt +measurement) OR recordAbstract: (+technical +debt +measurement)
- IEEE Xplore: (("Document Title":technical debt measurement) OR "Abstract": technical debt measurement)
- Google Scholar: "technical debt measurement"

The data have been extracted in two stages: in August 2018, when the initial version of the study started and in September 2019 to extend the study with the latest research available.

Only certain papers should be included to the final result: containing abstracts, considering TD as a main topic, written in English. No year constraint was specified, since we aimed at collecting all appropriate data despite of the date.

Many publications found in the digital libraries were not appropriate for our study since we were interested in primary studies published in referred workshops, conferences, and journals. Therefore, we excluded documents such as: summaries of workshops, tutorials, introductory descriptions of conferences, research plans, presentations, not primary studies, and technical reports. Therefore, we excluded all the documents that were not proper research papers.

Finally, we manually excluded all the papers not related to our research that passed the previous filters but still included in the list. The selection was performed after reading the entire content of the papers.

3 Results

We found 1,063 papers distributed as follows: ACM Digital Library (211), IEEE Xplore (317), and Google Scholar (535).

As expected, there was a significant overlap in the papers found in the different libraries. Therefore, the first step was merging the results and removing duplicates. Finally, at the end of the process, we selected 46 papers. The overall selection process is summarized in Fig. 1 (the numbers on the arrows show the amount of papers that passed each phase):

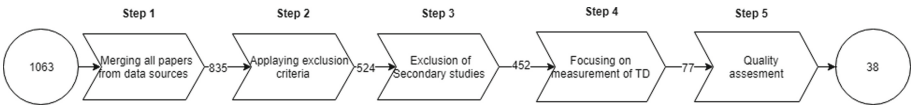

Fig. 1. Steps of the selection process.

- **Step 1: Merging All Papers from Data Sources.** The initial list included 1,063 papers but many duplicates were present. The identification of the duplicates was performed manually to avoid problems with minor character differences in the titles and in the author names. At the end, we had a list of 835 unique papers.
- **Step 2: Applying Exclusion Criteria.** At this stage, we applied the exclusion criteria resulting in a selection of 524 papers. At this stage we still kept in the list the secondary studies.
- **Step 3: Excluding not Primary Studies.** At this stage, we identified the secondary studies (e.g., systematic reviews, systematic mappings, etc.) that were removed from the list and analyzed in Sect. 4. The secondary studies identified are 10 and the list is reduced to 452 papers.
- **Step 4: Considering Studies Related to Measurement of TD.** Reading the title and the abstract of the 452 papers, we identified the studies related to the measurement of TD. We identified 38 papers distributed between 2011 and 2019 as described in Fig. 2.
- **Step 5: Quality Assessment.** We read the 77 papers identified and we excluded 39 of them since they were not dealing with the measurement of the technical debt even if from the title or the abstract they appeared appropriate for our investigation.

3.1 RQ1: Which Are the Existing Techniques for Measuring TD

The identified studies have been analyzed in terms of proposed techniques, their requirements about input data needed for the calculation of TD, the resulting information, advantages and disadvantages of the approach. Table 5 summarises

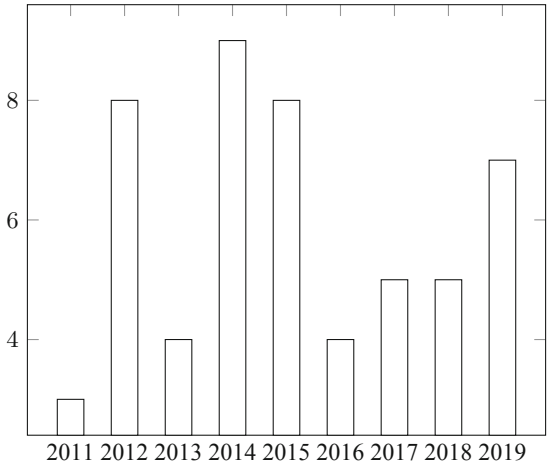

Fig. 2. Distribution of papers related to TD measurement over the years.

Table 1. Input of TD measurement techniques.

Technique (method)	Target quality level	Debt-estimating model	Number of should-fix violations	The hours to fix each violation	The cost of labor	Source code	Output data from static code analyzers	Candidate cloud-based mobile service	Past changes in the history of the system	Developer activity data
SQALE	✓	✓	-	-	-	-	-	-	-	-
CAST	-	-	✓	✓	✓	-	-	-	-	-
SIG	✓	-	-	-	✓	✓	-	-	-	-
A benchmarking-based model	✓	-	-	-	✓	✓	✓	-	-	-
A fluctuation-based modelling approach	-	-	-	-	-	-	-	✓	-	-
Breaking Point for TD	-	-	✓	✓	✓	-	-	-	✓	-
LOC and Fan-In to Quantify the Interest of SATD	-	-	-	-	-	✓	-	-	-	-
A framework for design level TD	-	-	-	-	-	✓	-	-	-	-
A framework for estimating interest on TD	-	-	-	-	-	-	-	-	-	✓
Modularity metrics for ATD	-	-	-	-	-	✓	-	-	✓	-
Detecting and quantifying SATD	-	-	-	-	-	✓	-	-	-	-
Pre-trained word embedding word2vec model	-	-	-	-	-	✓	-	-	-	-
Code metrics for TD	-	-	-	-	-	✓	-	-	-	-
Convolutional Neural Network	-	-	-	-	-	✓	-	-	-	-

the techniques identified while Table 1 compares the input required by the different techniques and Table 2 the output generated.

Letouzey [40] proposed a method for TD evaluation named Software Quality Assessment Based on Lifecycle Expectations (SQALE), which is described as an answer to the need for an objective and standardized open-source method with low false positives. At the official website of the method[1], there is a list of several tools able to analyze the code written in different languages.

The method defines how to formulate and organize non-functional requirements that can affect code quality defining a hierarchical structure of characteristics and sub-characteristics similar to the ISO quality model. SQALE has been developed to be automated and considers several properties of the code but two main aspects are not taken into account. The first one is that non-conformities

[1] http://www.sqale.org/.

for business or operations are not considered important by any index of SQALE (considering version 1.0 [41]). The second one is that there is no definition of the level of implementation of the requirements.

CAST [23] presents a formula with flexible parameters to measure TD. That flexibility implies the possibility of adjusting the parameters to the specificity of a particular organization. The approach defines five Health Factors that have a different impact on the overall TD: Changeability (30%), Transferability (40%), Robustness (18%), Security (7%), Performance Efficiency (5%).

Violations in each area are rated according to their severity and a formula is applied for calculating the final value of the debt. The approach has been evaluated on 745 business applications containing more than 10 KLOC using the CAST proprietary Application Intelligence Platform.

The SIG/TUViT approach [52] is based on a sound and quantitative approach for measuring software quality from source code. Moreover, the estimation of TD is based on empirical data using a model that is quite simple.

Mayr *et al.* [49] define a model that provides a combination of the benefits of the flexible approaches to quality changes and the simplicity of the SIG model. The approach requires only information from static code analysis. The output is simple as well, being the hours of work required to pay the debt.

Skourletopoulos *et al.* [68] developed a fluctuation-based modelling approach to TD. It measures the amount of profit not earned due to the under-usage of a given service and considering the probability of over-usage of the selected service that would lead to accumulated TD. The hypothesis is that service capacity affects to service choice, which is made with respect to the predicted fluctuations in the number of users over some time and the way TD is gradually paid off. Consequently, formulas for predicting appearance of TD were developed, as well as tools for validating them.

Chatzigeorgiou *et al.* [18] provide an estimation of a breaking point, that is when debt becomes too large to be paid off. The source code is initially assessed by fitness function based on the Entity Placement metric quantifying coupling and cohesion. The approach is based on the identification of the best design for a system. The cost of reaching that best system with necessary refactorings is calculated as well as number of versions leading to the breaking point. However, the authors point out some issues to be considered:

- only coupling and cohesion dimensions exist for the method, but TD has many other aspects
- maintenance effort means not just adding lines of code, but deleting and modifying them
- future maintenance effort cannot be predicted solely on the basis of past maintenance tasks

Kamei *et al.* [32] propose measuring the self-admitted TD interest with code metrics like LOC (because it well correlates with code complexity metrics) and Fan-In (showing how much one piece of code affects another one). They have validated the approach on the Apache JMeter project.

Marinescu [46] proposes a framework exploring TD symptoms at design level. The construction of such framework includes four steps:

1. definition of the principles for finding design defects
2. identification of a set of relevant design defects
3. estimation of the impact of each defect
4. the overall design quality is calculated

The framework also includes:

- a coarse-grain approach to monitor the evolution of TD over time
- a more detailed approach that enables locating and understanding individual flaws, which can lead to a systematic refactoring

The approach has been applied in a case study including 63 releases of two well known Eclipse projects (JDT and EMF). However, the conclusions of the case study cannot be generalized, considering the restricted number of systems analyzed and the limited number of design flaws that were included in the actual instantiation of the framework.

In the framework proposed by Singh *et al.* [66], TD estimation is based on measures of code maintainability obtained via static analysis and interest estimation based on activity data obtained by monitoring developer actions in the IDE. Main contribution of the framework is the integration of a developer activity data with code metrics and to improve the understanding of developer comprehension effort resulting in an improved accuracy of the estimation.

Although the Architectural Technical Debt (ATD) is difficult to measure, the Average Number of Modified Components per Commit (ANMCC) is a metric proposed in [43]. However, commit records may not exist anymore, therefore the authors suggest to use Index of Package Changing Impact (IPCI) and Index of Package Goal Focus (IPGF) instead of ANMCC. The advantage of using such two new metrics is the possibility of obtaining them directly from the source code. Then validation of correlation of that metrics with ANMCC is performed. However, the weakness of whole study is relying only on results of projects developed in C#.

Martini *et al.* [47] conducted a multiple embedded case study in seven sites at five large companies to investigate the current causes for the accumulation of Architectural TD (ATD). The authors investigated two research questions: (i) factors cause the accumulation of ATD, and (ii) current trends in practice in the accumulation and recovery of ATD over time. The authors provided a taxonomy of causes and their influence in the accumulation of ATD.

Maldonado *et al.* [45] examined code comments to identify and evaluate Self-admitted Architectural Debt (SATD). The strength of the approach is the usage of heuristics to eliminate comments which are not likely to affect TD. In addition, the method classify comments to different types of SATD.

Besker *et al.* [12] show critical results for SATD management, such as the fact that monitoring and evaluating ATD using accurate metrics is a key issue and it is not fully supported by any currently available tool.

Table 2. Output of TD measurement techniques.

Technique (method)	Design symptoms of TD	Remediation cost	Non-remediation cost	Relative amount of TD	Breaking point	Number of comments
SQALE	✓	–	–	–	–	–
CAST	–	✓	–	–	–	–
SIG	–	✓	✓	–	–	–
A benchmarking-based model	–	✓	–	–	–	–
A fluctuation-based modelling approach	–	–	–	✓	–	–
Breaking Point for TD	–	✓	✓	–	✓	–
LOC and Fan-In to Quantify the Interest of SATD	–	–	✓	–	–	–
A framework for design level TD	✓	–	–	–	–	–
A framework for estimating interest on TD	–	–	✓	–	–	–
Modularity metrics for ATD	–	–	–	✓	–	–
Detecting and quantifying SATD	–	–	–	–	–	✓
Code metrics for TD	–	–	–	✓	–	✓
Convolutional Neural Network	–	–	–	✓	–	✓
Pre-trained word embedding word2vec model	–	–	–	–	–	✓

Flisar and Podgorelec [26] developed a new SATD identification method which takes advantage of a large corpus of unlabeled code comments. The proposed feature enhancement method was used with the three most common feature selection methods (CHI, IG, and MI) and three well-known text classification algorithms (NB, SVM, and ME). It was tested on ten open source projects achieving 82% of correct predictions of SATD. The proposed method seems to be a good candidate to be adopted in practice.

Lenarduzzi *et al.* [38] applied the SZZ algorithm to label the fault-inducing commits and used 8 machine Learning techniques: Linear Regression, Random Forest, Gradient Boost, Extra Trees, Decision Trees, Bagging, AdaBoost, and SVM to show that the accuracy of TD can be improved. Authors found that among the 202 violations defined for Java by SonarQube, only 26 have a relatively low fault-proneness.

Pecorelli *et al.* [56] have reported on a large-scale empirical comparison between five different balancing techniques for ML-based code smell detection. The results suggest that ML models relying on SMOTE (Synthetic Minority Over-sampling Technique) realize the best performance. However, its training phase is not always feasible in practice. Furthermore, avoiding balancing does not dramatically impact the performance. Existing data balancing techniques are therefore inadequate for code smell detection. This hinders the feasibility of the current ML-based approaches.

Capitan and Vogel-Heuser [16] proposed metrics for identifying TD which based on IEC 61311-3 programming languages and adapted for the languages and cycle processing (Halstead's and McCabe's as examples).

Kumar *et al.* [35] proposed a novel approach for identifying TD in service composition in SaaS cloud. The approach combines time series forecasting and a newly proposed TD model to estimate the future debt and utility in the service composition. Through a real world case study, they demonstrate that the approach can successfully identify both the good and bad debts, while producing satisfactory accuracy on estimating the TD in the service composition in SaaS cloud.

Ciolkowski *et al.* [19] developed a prototype and a prediction model for forecasting potential savings based on proposed refactoring of key drivers of TD identified by the machine-learning model.

Verdecchia *et al.* [76] presented a novel approach to identify ATD of Android apps based on architectural guidelines extraction and modeling, architecture reverse engineering, and compliance checking.

Lavazza *et al.* [36] proposed a formal and executable model that supports the simulation of various scenarios in time-boxed software development and maintenance processes. The model is usable to show the effects that TD have on relevant issues such as productivity and quality, depending on how TD is managed, with special reference on how much effort is dedicated to TD repayment and when such effort is allocated.

TD visualizations were designed to improve stakeholder communication to support the business decision-making process at different levels of the organization. [53] concluded that the TD visualization contributes to improve the communication in the decision-making processes associated with the software lifecycle.

[61] addresses the problem of SATD (Self-Admitted Technical Debt) classification using a Convolutional Neural Network, which takes as input the source code comments and predicts as output whether the comment is a SATD comment or not.

Tsintzira *et al.* [74] used an established method for quantifying TD, namely FITTED, to measure the TD of an industrial software product and compare it to the perception of the software engineers.

3.2 RQ2: Which Are the Tools that Support the Automation of the Measurement of TD?

TD measurement techniques often require a large number of input data that require a large amount of effort to be extracted. Therefore, tools are of paramount importance to support development teams in the integration of TD measurement in their daily work. Table 3 provides a summary of the available tools and the methodology they implement.

SonarQube [27] implements the SQALE method of TD evaluation. It is used for continuous inspection of code quality to perform automatic reviews with

Table 3. Tools able to support the automation of the measurement of TD.

Technique (method)	Ref	Tool	Tool URL	Open source
SQALE	[40]	SonarQube	https://www.sonarqube.org/	Yes
		MIND	https://sourceforge.net/projects/mindyourdebt/	Yes
		FindBugs	http://findbugs.sourceforge.net/	Yes
Breaking Point for TD	[18]	JCaliper	http://se.uom.gr/index.php/projects/jcaliper/	Yes
A framework for design level TD	[46]	inFusion	https://chocolatey.org/packages/infusion/	No
A framework for estimating interest on TD	[66]	Blaze monitoring tool	https://sites.google.com/site/blazedemosite/home/about	No
A framework for the prioritization of technical	[2]	Tracy	–	No
A tool for auto identification and interactive monitoring of the evolution TD	[50]	VisminerTD	https://visminer.github.io	No
A tool for managing TD	[55]	DeepSourse	https://deepsource.io/	Yes
A tool to prevent mostly invisible technical debt	[7]	Debtgrep	–	No
A tool for automatic architectural smells detection for C/C++ projects	[13]	Arcan tool	http://www.essere.disco.unimib.it/wiki/arcan/	Yes
A tool calculates the presence of a set of code smells and calculates an Intensity index	[38]	JCodeOdor	http://www.essere.disco.unimib.it/jcodeodor/	Yes
A tool for detecting code smells from Java code and prioritizing technical debt based on the smells	[38]	JSpiRIT	–	No
A domain specific static code analysis tool	[14]	PLC software	–	No
A Tool for the strategic planning of TD in Agile Software Projects	[19]	ProDebt	–	No
A programming environment	[70]	EXA2PRO	–	No
Modularity metrics for ATD	[43]	TortoiseSVN	https://tortoisesvn.net/	Yes
LOC and Fan-In to Quantify the Interest of SATD	[32]	Understand	https://scitools.com/	No
		JDeodorant	https://github.com/tsantalis/JDeodorant	Yes
Detecting and quantifying SATD	[45]	SLOCCount	https://www.dwheeler.com/sloccount/sloccount.html	Yes

static analysis of code to detect bugs, code smells and security vulnerabilities in several programming languages.

MIND (ManagIng techNical Debt) is an open source tool which is, to the best of our knowledge, the first tool supporting the quantification and visualization of the interest [24]. Basically, it is a plug-in for SonarQube. MIND uses a few metrics to count the interest:

- Defect Proneness
- Maximum Defects per 100 LOC Touched
- Extra Defect Proneness
- Maximum Extra Defects per 100 LOC Touched

- Relative Extra Defect Proneness
- Average Relative Extra Defect Proneness
- Violation Density
- Linkage
- Estimation Error

JCaliper [18] was designed to find the placement of entities that minimizes the Entity Placement metric as a search-space exploration problem. It automatically extracts the number, type and sequence of refactoring activities required to obtain the design without TD.

Blaze is a monitoring tool [69] recording temporal sequence of developer actions, including code navigation actions and edit actions. The log produced is subsequently analysed to figure out class relationships and effort spent by a developer to understand program elements.

TortoiseSVN allows extracting commit records from standard SVN servers and any code repositories supporting Subversion, such as GitHub. That records are used by Li *et al.* [43] to perform ANMCC metric checking.

JDeodorant [73] is used in [32] for performing source code parsing. In particular, the ability to extract a comment and map it to its corresponding method is interesting. Later in the paper, to calculate the interest that is incurred over time, 16 code metrics were extracted using the Understand tool [1]. JDeodorant [73] is also used in [45] to parse the source code and extract the code comments. However, before that, the SLOCCount tool [77] is applied to calculate SLOC in Java files.

EXA2PRO [70] is a programming environment which integrates a set of tools and methodologies that allow to systematically address many exascale computing challenges, including performance, portability, programmability, abstraction and reusability, fault tolerance, and TD.

[60] presented a process framework for managing TD in commercial software product development. The framework integrates processes required for TD management with existing software quality management processes prescribed by the project management body of knowledge (PMBOK) (https://www.pmi.org/pmbok-guide-standards), and organizes the different processes for TD management in three steps: (1) make TD visible, (2) perform cost-benefit analysis, and (3) control TD. To implement the processes, they introduced a new artifact, the TD register, which stores the principal and the associated interest estimated for the TD related to an asset.

[7] introduced debtgrep, a tool to prevent from growing dependency violations, violation of naming conventions, usage of deprecated API's, and other kinds of mostly invisible TD. They provide some specific examples of use cases for debtgrep.

[67] introduced a tool used for extracting coupling and cohesion metrics at package level to study their impact on TD. The dataset of their study consisted of approximately 1,200 software packages.

[19] introduced the ProDebt tool, a methodology and a software tool to support the strategic planning of TD in the context of agile software development.

[6] proposed a new index for the evaluation of architectural issues as Architectural Smells (AS) and developed a tool to detect AS in Java projects. They focused on AS based on dependency issues, since components that are highly coupled and with a high number of dependencies cost more to maintain and can be considered more critical.

[13] introduces an open source tool for automatic architectural smells detection for C/C++ projects, by creating an abstraction of the project and defining the concept of dependency between elements belonging to the project in order to identify architectural smells.

[48] developed a holistic framework for the semi-automated identification and estimation of ATD in the form of non-modularized components.

[14] presented a static code analysis tool and its usage for identification of TD in IEC 61131-3. The tool supports both bottom-up (study the metric values at individual module and convention violations) as well as top-down analysis (study the call graphs). In addition, the authors provide an extra analysis (horizontally) by making a comparison on the metric results between different demonstrators.

[2] presented Tracy, a decision-making framework that prioritizes TD considering how IT assets support a company's business processes, thus providing a new perspective on TD management.

[50] presented VisminerTD, a tool that allows the automatic identification and interactive monitoring of the evolution of TD items by combining software metrics, code comment analysis, and information visualization. The results provided evidence on the use of the proposed tool, indicating (i) that it can be useful in supporting TD identification and TD monitoring activities and (ii) that it can bring gains in terms of comprehensiveness and efficacy when evaluating the desirable time to identify and monitor different types of debt.

[71] found that the tools used cannot help in identifying many important TD types, involving humans is necessary. Tools could help to identify TD faster or more accurately however, project priorities and current development activities are important to be considered together, along with the values of principal and interest, when deciding to provide a comprehensive evaluation of TD and pay it off.

3.3 RQ3: Are There Any Empirical Studies Able to Demonstrate the Usefulness of the Identified Techniques?

The empirical studies performed to validate the identified techniques are summarized in Table 4.

[28] assessed three methods [23,40,46] to find out if they effectively describe the relationship between the quality of the system and the level of TD.

Izurieta *et al.* [31] uses Nugroho *et al.* [52] to exemplify the methodology.

A Benchmarking-based Model of Mayr *et al.* [49] is closely related to their earlier work on benchmarking-oriented quality assessments. Also it calculates the remediation cost in a way similar to the approach of CAST [23].

Table 4. Identified techniques and the related empirical studies.

Technique (method)	Based on	Ref	Empirical study
SQALE	Previous version [40]	[41]	[28]
CAST	–	[23]	[28]
SIG	SIG quality model [30]	[52]	[31]
A Benchmarking-based Model	Benchmarking-oriented method [29], CAST [23]	[49]	[49]
A Fluctuation-Based Modelling Approach	–	[68]	[68]
Breaking Point for TD	CAST [23], previous version [4]	[18]	[18]
LOC and Fan-In to Quantify the Interest of SATD	–	[32]	–
A framework for design level TD	–	[46]	[28, 46]
A framework for estimating interest on TD	SIG [52]	[66]	[65]
Modularity metrics for ATD	–	[43]	[43]
Detecting and quantifying SATD	Previous version [59]	[45]	–
Convolutional Neural Network	–	[61]	–
Pre-trained word embedding word2vec model	–	[26]	–
A Prodebt Method for the strategic planning of TD in Agile Software Projects	–	[19]	–

Relevant code structure metrics in the framework for estimating interest on TD [66] were selected in such a way that related to maintainability and TD in [52]. Similar to the prior work, static code metrics are used.

[39] conducted an empirical study on 21 well-known mature open-source projects to confirm the hypothesis about the fault-proneness of the SonarQube violations.

[78] selected four different TD identification techniques (code smells, automatic static analysis (ASA) issues, grime buildup, and modularity violations) and applied them to 13 versions of the Apache Hadoop open source software project. The authros showed that different TD techniques are loosely coupled and therefore indicate problems in different locations of the source code. Moreover, their proxy interest indicators (change and defect-proneness) correlate with only a small subset of TD indicators.

[64] surveyed empirical research work in the arising topic of SATD after 2014 and until the compilation of this survey in July of 2018. They compiled the tools and datasets that can be used as a foundation to motivate and facilitate the submission of novel and improved approaches for managing and ultimately, repaying SATD. Simultaneously, authors observed a lack of studies focusing on the repayment and management of SATD, which is of critical importance.

3.4 RQ4: Are There Any Empirical Studies Able to Demonstrate the Usefulness of the Tools Identified?

In [54], TD was measured using two static code analysis tools (Findbugs [8] and SonarQube [27]). The goal was evaluating if the code produced with the Test Driven Development approach has a lower TD than code produced using other techniques. This two tools are widely used in the community for measuring TD.

Other studies tested SonarQube: [44] use it for measuring TD in a particle tracker system; [51] use it for several calculation of TD in the software supply chain; [15] describes a case study in Ericsson, where they had to observe TD measurement tools to use them for evaluation system creation based on ISO standard 15939:2007.

4 Related Work

Investigating the different approaches for measuring TD could be valuable to practitioners and researchers to provide a better understanding of the field and identify research gaps. However, we were not able to identify any secondary study related to the research questions we listed in Sect. 2. Instead, several others deal with TD in general.

The systematic mapping study of Li *et al.* [42] was initiated to find and analyze publications between 1992 and 2013 of TD and its management. After the selection of 92 studies authors classified 10 TD definition, identified 8 TD management activities, and collected 29 tools for the latter.

Another systematic mapping study of TD definitions, Poliakov [58] has performed full review of 159 papers. 107 definitions were separated into keywords. Consequently, the main achievement of the research is built keyword map, supplemented by synonyms and types of TD.

Another literature review has been done by Alves *et al.* [3] based on three research questions. They evaluated 100 studies of 2010–2014 and proposed initial taxonomy of TD types, list of indicators for identifying TD, and existing management strategies.

There is a study considering another aspect of the phenomenon. Ribeiro *et al.* [62] state that the evaluation of appropriate time to pay TD and applying an effective decision-making criteria are an important management goals. Consequently, authors identified 14 such criteria for development teams. Also the results showed gaps where further research can be performed.

Recently, Behutiye *et al.* [9] considered a narrow field of study related to TD, which means that they synthesized the state of the art of TD and its causes, consequences, and management strategies only in the context of agile software development (ASD). In particular, after processing systematic literature review 38 primary studies, out of 346 studies, were identified and analyzed. Then five research areas of interest related to the literature of TD in ASD, as well as 12 strategies for managing it have been found. Authors identified eight categories regarding the causes and five categories regarding the consequences of incurring TD in ASD.

In the case of work performed by Besker *et al.* [11] ATD is considered as affecting to system success and able to cause expensive repercussions, so the goal is to create new knowledge with interest in ATD. Research efforts should be synthesized and compiled for that. The main contributing outcome of the paper is a presentation of a novel descriptive model, providing comprehensive interpretation of ATD phenomenon.

Finally, the last related work focuses on a specific view of TD. Employing a method for syntactic literature review and applying it to seven digital library studies sources Ampatzoglou *et al.* (2016) [5] analyzed financial aspect of TD. Authors conclude that the communication between technical managers and project managers is beneficial, because a vocabulary will be provided, and high-quality goals will be set up. In order to achieve this, they introduced a glossary of terms and a classification scheme for financial approaches.

[63] investigates current state of TD based on 13 secondary studies, dated from 2012 to March 2018, the work shows several interesting conclusions such as coverage of areas (code, test, process, etc.).

[75] investigated the state-of-the-art and examined the major contributions that have been made in the field of TD estimation and forecasting. The authors stated that already existing methods and tools for TD estimation have not reached a satisfactory level of maturity yet, while there is still a large volume of potential metrics and techniques that have not been used and that could potentially increase the completeness of the TD estimation concept. In addition, although there has been extensive research with respect to predicting the evolution of individual software features, quality attributes, and quality properties that are directly or indirectly related to the TD of a software project, no concrete contributions exist in the related literature regarding TD forecasting.

[48] proposed a Strategic Adoption Model for Tracking Technical Debt (SAMTTD) aimed at helping companies to assess their TD management process and make decisions on its improvement.

[10] performed a systematic mapping study to identify and analyzed the empirical studies about TD between 2014 and 2017. The authors presented the most common indicators to identify and evaluate TD and identified thirteen types of TD. They identified forty-eight tools from the selected empirical studies and found that in some empirical studies, there are more than three tools used to investigate TD. Others develop new tools and compared their results to open tools; Also they paid special attention to SATD throughout the code comments and smells as the most applied as indicators of TD.

5 Threats to Validity

The main threats to validity identified are the following:

– Although the applied guideline [34] recommends to consider about seven digital libraries for performing an exhaustive search, in our case only three have been chosen. The reason of it is that other sources contain very few unique

papers compared to the ACM and IEEE digital libraries. Moreover, to avoid missing important papers we used Google Scholar that index almost everything.
- Constructing appropriate search string is a tricky task, since the title of some studies we are interested in does not include our key words, we decided to extend the search to the abstracts. Since we are interested in studies focusing on TD, we suppose that the key word is mentioned in the abstract.
- A way of automatically merging the outcome lists from that libraries is risky, since even a single different symbol in title might affect the result. For that reason, duplicates were identified and eliminated manually during the creation of a merged list.
- It may happen that some information has not been considered in our study since some papers could have been accidentally skipped or not present at the time of the query (September 2019).

6 Conclusions and Future Work

TD is a widely used buzzword but having a clear understanding of the available approaches and tools is quite difficult due to the large amount of material spread across a number of sources. This paper aimed at providing to researchers and practitioners an overview of the state of the art about TD focusing on the automated approaches.

According to the review, the research area is new and very active but still not mature. There is a constant presence of new approaches and tools that are not based on the outcomes of previous studies and researchers focus on validating their own approaches without any independent assessments. Moreover, such validations are frequently not replicable due to the usage of proprietary datasets. Therefore, additional effort is needed to identify cross-validated approaches with clear indications about their applicability. This is very important especially for practitioners since it is difficult for them to identify the models to apply in their specific contexts.

The study has also pointed out that where tools are available to support some specific approaches, they are often difficult to use requiring a complex setup and providing a limited support for the wide range of programming languages used in real projects. Moreover, most of the available tools are not able to measure or estimate the overall TD. They usually focus on the *remediation costs* and do not take into consideration the related interests (often named *non-remediation costs*) that are often very important for planning the development process and keep the debt under control over the entire lifecycle of a product.

Appendix

Table 5. Techniques with input, output, and calculation.

Technique	Input	Calculation	Output	Ref
SQALE	1. Target quality level (a list of nonfunctional requirements that define right code) 2. Debt-estimating model (associate each requirement with remediation function turning number of noncompliances into a remediation cost)	Run the code through the analysis tools and use remediation functions to work out remediation costs for each element. TD is the sum of remediation costs for all noncompliances. This debt is called the SQALE quality index (SQI)	Design symptoms of TD (Pyramid - an indicator to represent the specific distribution of TD for eight characteristics)	[41]
CAST	1. Number of should-fix violations in an application 2. The hours to fix each violation 3. The cost of labor	(($\sum$ high-severity violations) x (percentage to be fixed) x (average hours needed to fix) x ($ per hour)) + (($\sum$ medium-severity violations) x (percentage to be fixed) x (average hours needed to fix) x ($ per hour)) + (($\sum$ low-severity violations) x (percentage to be fixed) x (average hours needed to fix) x ($ per hour))	Remediation Cost	[23]
SIG	1. Source code 2. Target quality level 3. The cost of labor	For the extraction of measurement values from source code, the Software Analysis Toolkit of SIG is used. RE = RF * (SS * TF) * RA ME = $\frac{MF * (SS * (1+r)^t * TF)}{2^{(QualityLevel-3)/2}}$	1. Remediation Cost 2. Non-remediation Cost	[52]
A Benchmarking-based Model	1. Static code analyzers output data (reference projects) 2. Source code 3. Target quality level 4. The cost of labor	Tool support is available [57] that facilitates triggering code analysis tools as well as building the benchmark database and benchmark suite 1. the target quality level is specified 2. # of maximum allowed violations is calculated 3. # of violations to be fixed is calculated 4. # of violations to be fixed * the estimated effort for fixing * an hourly cost rate	Remediation Cost	[49]
A Fluctuation-based Modelling Approach	Candidate cloud-based mobile service	Quantifying the TD during the first year $TD_1 = 12 * [ppm * (U_{max} - U_{curr}) - C_{u/m} * (U_{max} - U_{curr})] = 12 * (U_{max} - U_{curr}) * (ppm - C_{u/m})$ from the second and onwards $TD_i = 12 * [K_{i-2} * [U_{max} - L_{i-2}] - M_{i-2} * [U_{max} - L_{i-2}]] = 12 * (U_{max} - L_{i-2}) * (K_{i-2} - M_{i-2})$, i > 1	Relative amount of TD	[68]
Breaking Point for TD	1. Number of should-fix violations in an application 2. The hours to fix each violation 3. The cost of labor 4. Past changes in the history of the system (LOC)	TD-Principal is calculated as a function of first 3 input variables. $Interest = addedLOC * (1 - \frac{FitnessValue(optimum)}{FitnessValue(actual)})$ $versions = \frac{Principal(\$)}{Interest(\$)}$	1. Remediation Cost 2. Non-remediation Cost 3. Breaking point	[18]

(*continued*)

Table 5. (*continued*)

Technique	Input	Calculation	Output	Ref
LOC and Fan-In to Quantify the Interest of SATD	Source code	1. Extracting comments and mapping them to its corresponding methods 2. Determination of the change over time in these SATD methods 3. Determining metrics measuring interest 4. Calculating the interest per SATD instance	Non-remediation Cost	[32]
A framework for design level TD	Source code	1. Select a set of relevant design flaws 2. Define rules for the detection of each design flaw 3. Measure the negative influence of each detected flaw instance $FlawImpactScore(FIS)_{flaw\_instance} = I_{flaw\_type} * G_{flaw\_type} * S_{flaw\_instance}$ 4. Compute an overall score $DebtSymptomsIndex = \frac{\sum FIS_{flaw\_instance}}{KLOC}$	Design symptoms of TD	[46]
A framework for estimating interest on TD	Developer activity data	1. Establishing sessions 2. Calculate metrics related to comprehension effort within a session 3. $Interest(I) = I_{current} - I_{ideal}$ Static metrics show presence of TD in classes Comprehension effort metrics quantify effort to comprehend the classes	Non-remediation Cost	[66]
Modularity metrics for ATD	Past changes in the history of the system (commit records) or Source code	1. Parse the commit records to extract needed data items for ANMCC calculation 2. Filtering out data in commit records 3. $ANMCC = (\sum_{j=1}^{h} NMC(k+j))/h$ A higher ANMCC entails potential increase in ATD or 1. Code map generation (XML) 2. Code map parsing 3. Modularity metrics calculation A higher IPCI or IPGF indicate less ATD	Relative amount of TD	[43]
Detecting and quantifying SATD	Source code	1. Project Data Extraction (release used, the number of classes, the total source lines of code, the total extracted comments and the number of contributors) 2. Parsing the source code and extracting the code comments 3. Filtering comments 4. Manual classification into five different types of SATD	# of comments (number of individual line, block, and Javadoc comments)	[45]
Pre-trained word embedding word2vec model	Source code	The framework is composed of three separated phases: The word embedding phase, the model training phase, and the prediction phase. In the word embedding phase, a word2vec model is built to support the feature enhancement method for the model training phase. In the model training phase, the SATD classifier is built within the next four steps: Comments' prepossessing, feature selection, feature enhancement, and classifier training. In the prediction phase, the built SATD classification model can be used to perform a SATD prediction upon new, previously unseen code comments	# of SATD comments	[26]

(*continued*)

Table 5. (*continued*)

Technique	Input	Calculation	Output	Ref
Code metrics for Technical Debt	Source code	1. Parsing the source code and extracting the code metrics 2. Based on source code metrics evaluate existence of TD	Using the metrics and proposed in the work rules it could be identified presence or absence of TD	[17]
Convolution Neural Network	Set of source code comments and their corresponding SATD/non-SATD labels	The approach includes four main phases: Model Training, SATD Prediction, Key Phrase Extraction, and SATD Pattern Identification. 1. The trained model is used to predict the SATD/non-SATD label given an unseen source code comment (prediction phase). 2. The trained model is de-convolutioned to extract SATD-indicating key phrases in the input code comments that contribute most to the model's classification decisions, which in turn are summarized into a set of intuitive SATD patterns.	# of comments that are SATD and not-SATD	[61]
A Prodebt method for the strategic planning of TD	Source code	1. Based on coding effort and changes to the source code Productivity are measuring by $P = \frac{CodingOutput}{EffortInvested}$ 2. A function introduced $f : q \rightarrow P$, where q is the vector of metrics in the quality model 3. Applied machine learning (a random forest algorithm) to approximate f 4. The saved effort/Approximated TD $e_x = e_o(1 - \frac{P_o}{P_x})$	Using the productivity approximator f they computed productivity for a hypothetical quality level, obtained by modifying one quality metric/percent and keeping all other quality metrics at the same level	[19]

References

1. Scientific toolworks, inc. understand 2.6. http://www.scitools.com/
2. de Almeida, R.R., Treude, C., Kulesza, U.: Tracy: a business-driven technical debt prioritization framework. In: 35th International Conference on Software Maintenance and Evolution (ICSME 2019) (2019)
3. Alves, N.S., Mendes, T.S., de Mendonça, M.G., Spínola, R.O., Shull, F., Seaman, C.: Identification and management of technical debt: a systematic mapping study. Inf. Softw. Technol. **70**, 100–121 (2016)
4. Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., Chatzigeorgiou, A.: Establishing a framework for managing interest in technical debt. In: 5th International Symposium on Business Modeling and Software Design, BMSD (2015)
5. Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., Avgeriou, P.: The financial aspect of managing technical debt: a systematic literature review. Inf. Softw. Technol. **64**, 52–73 (2015)
6. Arcelli-Fontana, F., Pigazzini, I., Roveda, R., Tamburri, D., Zanoni, M., Nitto, E.: ARCAN: a tool for architectural smells detection. In: Proceeding of the International Conference on Software Architecture, ICSA 2017, pp. 282–285. IEEE (2017)
7. Arvedahl, S.: Introducing debtgrep, a tool for fighting technical debt in base station software. In: Proceedings of the 2018 International Conference on Technical Debt TechDebt 2018, pp. 51–52. ACM (2018)

8. Ayewah, N., Hovemeyer, D., Morgenthaler, J.D., Penix, J., Pugh, W.: Using static analysis to find bugs. IEEE Softw. **25**(5), 22–29 (2008)
9. Behutiye, W.N., Rodríguez, P., Oivo, M., Tosun, A.: Analyzing the concept of technical debt in the context of agile software development: a systematic literature review. Inf. Softw. Technol. **82**, 139–158 (2017)
10. BenIdris, M., Ammar, H., Dzielski, D.: Investigate, identify and estimate the technical debt: a systematic mapping study. Int. J. Softw. Eng. Appl. **9**(5), 1–14 (2018)
11. Besker, T., Martini, A., Bosch, J.: A systematic literature review and a unified model of ATD. In: 2016 42th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 189–197. IEEE (2016)
12. Besker, T., Martini, A., Bosch, J.: Managing architectural technical debt: a unified model and systematic literature review. J. Syst. Softw. **135**, 1–16 (2018)
13. Biaggi, A., Arcelli Fontana, F., Roveda, R.: An architectural smells detection tool for C and C++ projects. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 417–420 (2018)
14. Bougouffa, S., Dong, Q.H., Diehm, S., Gemein, F., Vogel-Heuser, B.: Technical debt indication in plc code for automated production systems: Introducing a domain specific static code analysis tool. IFAC-Papers OnLine **51**(10), 70–75 (2018). 3rd IFAC Conference on Embedded Systems, Computational Intelligence and Telematics in Control CESCIT 2018
15. Britsman, E., Tanriverdi, Ö.: Identifying technical debt impact on maintenance effort-an industrial case study (2015). https://gupea.ub.gu.se/handle/2077/40110
16. Capitán, L., Vogel-Heuser, B.: Metrics for software quality in automated production systems as an indicator for technical debt. In: 2017 13th IEEE Conference on Automation Science and Engineering (CASE), pp. 709–716 (2017)
17. Capitán, L., Vogel-Heuser, B.: Metrics for software quality in automated production systems as an indicator for technical debt. In: 2017 13th IEEE Conference on Automation Science and Engineering (CASE), pp. 709–716, August 2017
18. Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A., Amanatidis, T.: Estimating the breaking point for technical debt. In: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), pp. 53–56. IEEE (2015)
19. Ciolkowski, M., Guzmán, L., Trendowicz, A., Salfner, F.: Lessons learned from the ProDebt research project on planning technical debt strategically. In: Felderer, M., Méndez Fernández, D., Turhan, B., Kalinowski, M., Sarro, F., Winkler, D. (eds.) PROFES 2017. LNCS, vol. 10611, pp. 523–534. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-69926-4_42
20. Coman, I.D., Sillitti, A., Succi, G.: Investigating the usefulness of pair-programming in a mature agile team. In: 9th International Conference on eXtreme Programming and Agile Processes in Software Engineering (XP2008), June 2008
21. Corral, L., Sillitti, A., Succi, G.: Software development processes for mobile systems: is agile really taking over the business? In: 1st International Workshop on Mobile-Enabled Systems (MOBS 2013) at ICSE 2013, June 2013
22. Cunningham, W.: The WyCash portfolio management system. In: Addendum to the Proceedings on Object-Oriented Programming Systems, Languages, and Applications (Addendum), OOPSLA 1992, Vancouver, British Columbia, Canada, pp. 29–30. Association for Computing Machinery, New York (1992). https://doi.org/10.1145/157709.157715
23. Curtis, B., Sappidi, J., Szynkarski, A.: Estimating the size, cost, and types of technical debt. In: Proceedings of the Third International Workshop on Managing Technical Debt, pp. 49–53. IEEE Press (2012)

24. Falessi, D., Reichel, A.: Towards an open-source tool for measuring and visualizing the interest of technical debt. In: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), pp. 1–8. IEEE (2015)
25. Falessi, D., Shaw, M.A., Shull, F., Mullen, K., Keymind, M.S.: Practical considerations, challenges, and requirements of tool-support for managing technical debt. In: 2013 4th International Workshop on Managing Technical Debt (MTD), pp. 16–19. IEEE (2013)
26. Flisar, J., Podgorelec, V.: Identification of self-admitted technical debt using enhanced feature selection based on word embedding. IEEE Access **7**, 106475–106494 (2019)
27. Gaudin, O.: Evaluate your technical debt with sonar. Sonar, June 2009
28. Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., Williams, B.: The correspondence between software quality models and technical debt estimation approaches. In: 2014 Sixth International Workshop on Managing Technical Debt (MTD), pp. 19–26. IEEE (2014)
29. Gruber, H., Plösch, R., Saft, M.: On the validity of benchmarking for evaluating code quality. In: IWSM/MENSURA 2010 (2010). https://www.iwsm-mensura.org/2010-conference/
30. Heitlager, I., Kuipers, T., Visser, J.: A practical model for measuring maintainability. In: 6th International Conference on the Quality of Information and Communications Technology, QUATIC 2007, pp. 30–39. IEEE (2007)
31. Izurieta, C., Griffith, I., Reimanis, D., Luhr, R.: On the uncertainty of technical debt measurements. In: 2013 International Conference on Information Science and Applications (ICISA), pp. 1–4. IEEE (2013)
32. Kamei, Y., Maldonado, E.D.S., Shihab, E., Ubayashi, N.: Using analytics to quantify interest of self-admitted technical debt. In: QuASoQ/TDA@ APSEC, pp. 68–71 (2016)
33. Khomyakov, I., Makhmutov, Z., Mirgalimova, R., Sillitti, A.: Automated measurement of technical debt: a systematic literature review. In: 21st International Conference on Enterprise Information Systems (ICEIS 2019), May 2019
34. Kitchenham, B., Charters, S.: Guidelines for performing systematic literature reviews in software engineering (version 2.3). Technical report, Keele University and University of Durham (2007)
35. Kumar, S., Bahsoon, R., Chen, T., Buyya, R.: Identifying and estimating technical debt for service composition in SaaS cloud. In: 2019 IEEE International Conference on Web Services (ICWS), pp. 121–125, July 2019
36. Lavazza, L., Morasca, S., Tosi, D.: A method to optimize technical debt management in timed-boxed processes. In: The Thirteenth International Conference on Software Engineering Advances (ICSEA 2018) (2018)
37. Lenarduzzi, V., Sillitti, A., Taibi, D.: Analyzing forty years of software maintenance models. In: 39th International Conference on Software Engineering (ICSE 2017), May 2017
38. Lenarduzzi, V., Martini, A., Taibi, D., Tamburri, D.A.: Towards surgically-precise technical debt estimation: early results and research roadmap. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE 2019, pp. 37–42. ACM (2019)
39. Lenarduzzi, V., Saarimäki, N., Taibi, D.: The technical debt dataset. In: Proceedings of the Fifteenth International Conference on Predictive Models and Data Analytics in Software Engineering PROMISE 2019, pp. 2–11. ACM (2019)

40. Letouzey, J.L.: The SQALE method for evaluating technical debt. In: 2012 Third International Workshop on Managing Technical Debt (MTD), pp. 31–36. IEEE (2012)
41. Letouzey, J.L., Ilkiewicz, M.: Managing technical debt with the SQALE method. IEEE Softw. **29**(6), 44–51 (2012)
42. Li, Z., Avgeriou, P., Liang, P.: A systematic mapping study on technical debt and its management. J. Syst. Softw. **101**, 193–220 (2015)
43. Li, Z., Liang, P., Avgeriou, P., Guelfi, N., Ampatzoglou, A.: An empirical investigation of modularity metrics for indicating architectural technical debt. In: Proceedings of the 10th international ACM Sigsoft conference on Quality of Software Architectures, pp. 119–128. ACM (2014)
44. Luhr, R.L., et al.: The application of technical debt mitigation techniques to a multidisciplinary software project. Ph.D. thesis, Montana State University-Bozeman, College of Engineering (2015)
45. Maldonado, E.D.S., Shihab, E.: Detecting and quantifying different types of self-admitted technical debt. In: 2015 IEEE 7th International Workshop on Managing Technical Debt (MTD), pp. 9–15. IEEE (2015)
46. Marinescu, R.: Assessing technical debt by identifying design flaws in software systems. IBM J. Res. Dev. **56**(5), 9–1 (2012)
47. Martini, A., Bosch, J., Chaudron, M.: Architecture technical debt: understanding causes and a qualitative model. In: Proceedings of the 2014 40th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2014, pp. 85–92. IEEE (2014)
48. Martini, A., Sikander, E., Madlani, N.: A semi-automated framework for the identification and estimation of architectural technical debt: a comparative case-study on the modularization of a software component. Inf. Softw. Technol. **93**, 264–279 (2018)
49. Mayr, A., Plösch, R., Körner, C.: A benchmarking-based model for technical debt calculation. In: 2014 14th International Conference on Quality Software (QSIC), pp. 305–314. IEEE (2014)
50. Mendes, T.S., Gomes, F.G.S., Gonçalves, D.P., Mendonça, M.G., Novais, R.L., Spínola, R.O.: VisminerTD: a tool for automatic identification and interactive monitoring of the evolution of technical debt items. J. Brazil. Comput. Soc. **25**(1), 2 (2019)
51. Monteith, J.Y., McGregor, J.D.: Exploring software supply chains from a technical debt perspective. In: Proceedings of the 4th International Workshop on Managing Technical Debt, pp. 32–38. IEEE Press (2013)
52. Nugroho, A., Visser, J., Kuipers, T.: An empirical model of technical debt and interest. In: Proceedings of the 2nd Workshop on Managing Technical Debt, pp. 1–8. ACM (2011)
53. Pacheco, A., Marín-Raventós, G., López, G.: Designing a technical debt visualization tool to improve stakeholder communication in the decision-making process: a case study. In: Tjoa, A.M., Raffai, M., Doucek, P., Novak, N.M. (eds.) CONFENIS 2018. LNBIP, vol. 327, pp. 15–26. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99040-8_2
54. Parodi, E., Matalonga, S., Macchi, D., Solari, M.: Comparing technical debt in student exercises using test driven development, test last and ad hoc programming. In: 2016 XLII Latin American Computing Conference (CLEI), pp. 1–10. IEEE (2016)
55. Parthiban, D.G.: Examination of tools for managing different dimensions of technical debt. CoRR abs/1904.11062 (2019). http://arxiv.org/abs/1904.11062

56. Pecorelli, F., Di Nucci, D., De Roover, C., De Lucia, A.: On the role of data balancing for machine learning-based code smell detection. In: Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation, MaLTeSQuE 2019, pp. 19–24. ACM (2019)
57. Ploesch, R., Gruber, H., Pomberger, G., Saft, M., Schiffer, S.: Tool support for expert-centred code assessments. In: 2008 1st International Conference on Software Testing, Verification, and Validation, pp. 258–267. IEEE (2008)
58. Poliakov, D., et al.: A systematic mapping study on technical debt definition (2015). https://pdfs.semanticscholar.org/a425/2d6a5d9522a85984379f8ee3bf118df782c1.pdf
59. Potdar, A., Shihab, E.: An exploratory study on self-admitted technical debt. In: 2014 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 91–100. IEEE (2014)
60. Ramasubbu, N., Kemerer, C.F.: Integrating technical debt management and software quality management processes: a normative framework and field tests. IEEE Trans. Software Eng. **45**(3), 285–300 (2019)
61. Ren, X., Xing, Z., Xia, X., Lo, D., Wang, X., Grundy, J.: Neural network-based detection of self-admitted technical debt: from performance to explainability. ACM Trans. Softw. Eng. Methodol. **28**(3), 15:1–15:45 (2019)
62. Ribeiro, L.F., de Freitas Farias, M.A., Mendonça, M.G., Spínola, R.O.: Decision criteria for the payment of technical debt in software projects: a systematic mapping study. In: ICEIS, no. 1, pp. 572–579 (2016)
63. Rios, N., de Mendonça Neto, M.G., Spínola, R.O.: A tertiary study on technical debt: types, management strategies, research trends, and base information for practitioners. Inf. Softw. Technol. **102**, 117–145 (2018)
64. Sierra, G., Shihab, E., Kamei, Y.: A survey of self-admitted technical debt. J. Syst. Softw. **152**, 70–82 (2019)
65. Singh, V., Pollock, L.L., Snipes, W., Kraft, N.A.: A case study of program comprehension effort and technical debt estimations. In: 2016 IEEE 24th International Conference on Program Comprehension (ICPC), pp. 1–9. IEEE (2016)
66. Singh, V., Snipes, W., Kraft, N.A.: A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. In: 2014 Sixth International Workshop on Managing Technical Debt (MTD), pp. 27–30. IEEE (2014)
67. Skiada, P., Ampatzoglou, A., Arvanitou, E., Chatzigeorgiou, A., Stamelos, I.: Exploring the relationship between software modularity and technical debt. In: 2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA), pp. 404–407 (2018)
68. Skourletopoulos, G., Mavromoustakis, C.X., Mastorakis, G., Rodrigues, J.J., Chatzimisios, P., Batalla, J.M.: A fluctuation-based modelling approach to quantification of the technical debt on mobile cloud-based service level. In: 2015 IEEE Globecom Workshops (GC Wkshps), pp. 1–6. IEEE (2015)
69. Snipes, W., Nair, A.R., Murphy-Hill, E.: Experiences gamifying developer adoption of practices and tools. In: Companion Proceedings of the 36th International Conference on Software Engineering, pp. 105–114. ACM (2014)
70. Soudris, D., et al.: Exa2pro programming environment: architecture and applications. In: Proceedings of the 18th International Conference on Embedded Computer Systems: Architectures, Modeling, and Simulation, SAMOS 2018, pp. 202–209. ACM (2018)

71. Spínola, R.O., Zazworka, N., Vetro, A., Shull, F., Seaman, C.: Understanding automated and human-based technical debt identification approaches-a two-phase study. J. Brazil. Comput. Soc. **25**(1), 5 (2019)
72. Tom, E., Aurum, A., Vidgen, R.: An exploration of technical debt. J. Syst. Softw. **86**(6), 1498–1516 (2013)
73. Tsantalis, N., Chaikalis, T., Chatzigeorgiou, A.: Jdeodorant: identification and removal of type-checking bad smells. In: 12th European Conference on Software Maintenance and Reengineering, CSMR 2008, pp. 329–331. IEEE (2008)
74. Tsintzira, A.A., Ampatzoglou, A., Matei, O., Ampatzoglou, A., Chatzigeorgiou, A., Heb, R.: Technical debt quantification through metrics: an industrial validation. In: 15th China-Europe International Symposium on Software Engineering Education (CEISEE 2019). IEEE (2019)
75. Tsoukalas, D., Siavvas, M., Jankovic, M., Kehagias, D., Chatzigeorgiou, A., Tzovaras, D.: Methods and tools for td estimation and forecasting: a state-of-the-art survey. In: 2018 International Conference on Intelligent Systems (IS), pp. 698–705 (2018)
76. Verdecchia, R.: Identifying architectural technical debt in android applications through automated compliance checking. In: Proceedings of the 5th International Conference on Mobile Software Engineering and Systems, MOBILESoft 2018, pp. 35–36. ACM (2018)
77. Wheeler, D.A.: More than a gigabuck: estimating gnu/linux's size (2001). https://dwheeler.com/sloc/redhat71-v1/redhat71sloc.html
78. Zazworka, N., et al.: Comparing four approaches for technical debt identification. Software Qual. J. **22**(3), 403–426 (2014)