Automated Measurement of Technical Debt: A Systematic Literature Review

Ilya Khomyakov, Zufar Makhmutov, Ruzilya Mirgalimova and Alberto Sillitti
Innopolis University, Russian Federation

Keywords: Technical Debt, Measurement, Literature Review.

Abstract: *Background*: Technical Debt (TD) is a quite complex concept that includes several aspect of software development. Often, people talk about TD as the amount of postponed work but this is just a basic approximation of the concept that includes many aspects that are technical and managerial. If TD is managed properly, it can provide a huge advantage but it can also make projects unmaintainable, if not. Therefore, being able of measuring TD is a very important aspect for a proper management of the development process. However, due to the complexity of the concept and the different aspects that are involved, such measurement it not easy and there are several different approaches in literature.
Goals: This work aims at investigating the existing approaches to the measurement and the analysis of TD focusing on quantitative methods that could also be automated.
Method: The Systematic Literature Review (SLR) approach was applied to 331 studies obtained from the three largest digital libraries and databases.
Results: After applying all filtering stages, 21 papers out of 331 were selected and deeply analyzed. The majority of them suggested new approaches to measure TD using different criteria not built on top of existing ones.
Conclusions: Existing studies related to the measurement of TD were observed and analyzed. The findings have shown that the field is not mature and there are several models that have almost no independent validation. Moreover few tools for helping to automate the evaluation process exist.

1 INTRODUCTION

A keen competition among companies for customer satisfaction is one of the reasons behind the continuous pressure to produce high-quality and maintainable source code in continuously reduced timeframes (Kan, 2002). Several studies has been performed focusing on the activities of the developers (Coman and Sillitti, 2007) (Coman and Sillitti, 2008) (Moser et al., 2008) (Coman et al., 2014) and considering code quality as the main criterion for releasing a product could lead to consume an excessive amount of resources (Corral et al., 2014). However, this criterion is critical to achieve a high level of customer satisfaction and the quality of the product is often a prerequisite to achieve success. Consequently, companies focusing on the quality of their product usually have a better market success (Boehm et al., 2001).

There are situations in which it is required to reduce the development time to achieve a minimum working product. This is a typical situation of startup companies that have very aggressive schedules to deliver the product that allows them to survive. In such context, the sub-optimal decisions that decrease the quality of the system lead to a strategic TD (Tom et al., 2013). This allows the company to deliver the product for which the customer pays allowing the company to survive. In any case, companies should be aware that in the long run such sub-optimal decisions require additional effort in the future to fix the product (Coman et al., 2008) (Corral et al., 2013).

This phenomenon was originally described by Ward Cunningham in 1992 (Cunningham, 1992) 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 (Tom et al., 2013) (Lenarduzzi et al., 2017).

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

(Falessi et al., 2013). Being able to estimate TD allows development teams and the manager to plan the work properly.

It may also happen that TD is too high to be payed (Chatzigeorgiou et al., 2015), requiring different approaches to address it (e.g., rewriting the system). However, knowing that and how the system reach that condition could help in the identification of past mistakes and improve the development.

Moreover, the measurement of TD should be performed automatically to avoid increasing the load of the developers and being able to monitor that continuously. This is particularly useful in conjunction to the usage of Agile approaches that can use such information to adapt iterations continuously.

For all these reasons, being able of measuring automatically TD 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.

The paper is organized as follows: Section 2 describes the adopted methodology; Section 3 discusses the findings; Section 4 investigates the related work; Section 5 analyzes the threats to validity; finally, Section 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 (Kitchenham and Charters, 2007) for performing such reviews in the software engineering area.

The main goal of this work is to review 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, 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 word *technical debt*. Consequently, we used appropriate queries for each library. The data have been extracted in August 2018, when the study was started.

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. 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 603 papers distributed as follows: ACM Digital Library (111), IEEE Xplore (194), Google Scholar (298).

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 21 papers. The overall selection process is summarized in Figure 1 (the numbers on the arrows show the amount of papers that passed each phase):

- **Step 1: Merging all Papers from Data Sources.** The initial list included 603 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 331 unique papers.
- **Step 2: Applying Exclusion Criteria.** At this stage, we applied the exclusion criteria resulting in a selection of 274 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

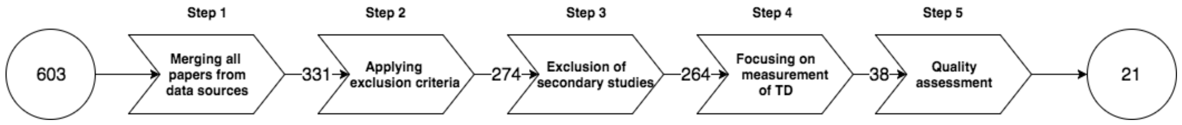
Step 1
Step 2
Step 3
Step 4
Step 5
603
Merging all papers from data sources
331
Applying exclusion criteria
274
Exclusion of secondary studies
264
Focusing on measurement of TD
38
Quality assessment
21

Figure 1: Steps of the selection process.

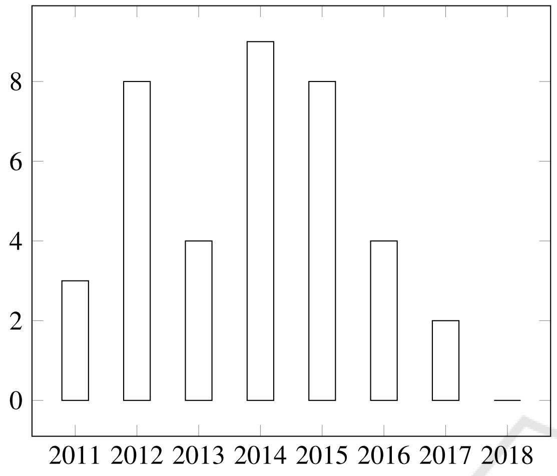

Figure 2: Distribution of papers related to TD measurement over the years.

Section 4. The secondary studies identified are 10 and the list is reduced to 264 papers.

- **Step 4: Considering Studies Related to Measurement of TD.** Reading the title and the abstract of the 264 papers, we identified the studies related to the measurement of TD. We identified 38 papers distributed between 2011 and 2018 as described in Figure 2.
- **Step 5: Quality Assessment.** We read the 38 papers identified and we excluded 17 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 the techniques identified while Table 1 compares the input required by the different techniques and Table 2 the output generated.

Letouzey (Letouzey, 2012) 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 herarchical 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 for business or operations are not considered important by any index of SQALE (considering version 1.0 (Letouzey and Ilkiewicz, 2012)). The second one is that there is no definition of the level of implementation of the requirements.

CAST (Curtis et al., 2012) 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 (Nugroho et al., 2011) 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 *at al.* (Mayr et al., 2014) 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.* (2015) (Skourletopoulos et al., 2015) 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

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

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	-	-	-	-	-	✓	-	-	-	-

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.* (2015) (Chatzigeorgiou et al., 2015) 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.* (Kamei et al., 2016) 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 (Marinescu, 2012) 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.* (Singh et al., 2014), 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 (Li et al., 2014). 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#.

Maldonado *et al.* (Maldonado and Shihab, 2015) examine 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 addi-

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	-	-	-	-	-	✓

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

Technique (method)	**Ref**	**Tool**	**Tool URL**	**Open source**
SQALE	(Letouzey, 2012)	SonarQube	https://www.sonarqube.org/	yes
		MIND	https://sourceforge.net/projects/mindyourdebt/	yes
		FindBugs	http://findbugs.sourceforge.net/	yes
Breaking Point for TD	(Chatzigeorgiou et al., 2015)	JCaliper	http://se.uom.gr/index.php/projects/jcaliper/	yes
A framework for design level TD	(Marinescu, 2012)	inFusion	https://chocolatey.org/packages/infusion/	no
A framework for estimating interest on TD	(Singh et al., 2014)	Blaze monitoring tool	https://sites.google.com/site/blazedemosite/home/about	no
Modularity metrics for ATD	(Li et al., 2014)	TortoiseSVN	https://tortoisesvn.net/	yes
LOC and Fan-In to Quantify the Interest of SATD	(Kamei et al., 2016)	Understand	https://scitools.com/	no
		JDeodorant	https://github.com/tsantalis/JDeodorant	yes
Detecting and quantifying SATD	(Maldonado and Shihab, 2015)	SLOCCount	https://www.dwheeler.com/sloccount/sloccount.html	yes

tion, the method classify comments to different types of SATD.

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 (Gaudin, 2009) implements the SQALE method of TD evaluation. It is used for continuous inspection of code quality to perform automatic reviews with 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 (Falessi and Reichel, 2015). 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 (Chatzigeorgiou et al., 2015) 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 (Snipes et al., 2014) 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.* (Li et al., 2014) to perform ANMCC metric checking.

JDeodorant (Tsantalis et al., 2008) is used in (Kamei et al., 2016) 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 (und,). JDeodorant (Tsantalis et al., 2008) is also used in (Maldonado and Shihab, 2015) to parse the source code and extract the code comments. However, before that, the SLOCCount tool (Wheeler, 2001) is applied to calculate SLOC in Java files.

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.

(Griffith et al., 2014) assessed three methods ((Letouzey, 2012) (Curtis et al., 2012) (Marinescu, 2012)) to find out if they effectively describe the relationship between the quality of the system and the level of TD.

Izurieta *et al.* (Izurieta et al., 2013) uses Nugroho *et al.* (Nugroho et al., 2011) to exemplify the methodology.

A Benchmarking-based Model of Mayr *et al.* (Mayr et al., 2014) 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 (Curtis et al., 2012).

Relevant code structure metrics in the framework for estimating interest on TD (Singh et al., 2014) were selected in such a way that related to maintainability and TD in (Nugroho et al., 2011). Similar to the prior work, static code metrics are used.

3.4 RQ4: Are there any Empirical Studies able to Demonstrate the Usefulness of the Tools Identified?

In (Parodi et al., 2016), TD was measured using two static code analysis tools (Findbugs (Ayewah et al., 2008) and SonarQube (Gaudin, 2009)). 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: (Luhr et al., 2015) use it for measuring TD in a particle tracker system; (Monteith and McGregor, 2013) use it for several calculation of TD in the software supply chain; (Britsman and Tanriverdi, 2015) 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 Section 2. Instead, several others deal with TD in general.

The systematic mapping study of Li *et al.* (Li et al., 2015) 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 (Poliakov et al., 2015) 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.* (Alves et al., 2016) 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.* (Ribeiro et al., 2016) 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.* (Behutiye et al., 2017) 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

Table 4: Identified techniques and the related empirical studies.

Technique (method)	**Based on**	**Ref**	**Empirical study**
SQALE	previous version (Letouzey, 2012)	(Letouzey and Ilkiewicz, 2012)	(Griffith et al., 2014)
CAST	-	(Curtis et al., 2012)	(Griffith et al., 2014)
SIG	SIG quality model (Heitlager et al., 2007)	(Nugroho et al., 2011)	(Izurieta et al., 2013)
A Benchmarking-based Model	benchmarking-oriented method (Gruber et al., 2010), CAST (Curtis et al., 2012)	(Mayr et al., 2014)	(Mayr et al., 2014)
A Fluctuation-Based Modelling Approach	-	(Skourletopoulos et al., 2015)	(Skourletopoulos et al., 2015)
Breaking Point for TD	CAST (Curtis et al., 2012), previous version (Ampatzoglou et al., 2015a)	(Chatzigeorgiou et al., 2015)	(Chatzigeorgiou et al., 2015)
LOC and Fan-In to Quantify the Interest of SATD	-	(Kamei et al., 2016)	-
A framework for design level TD	-	(Marinescu, 2012)	(Marinescu, 2012), (Griffith et al., 2014)
A framework for estimating interest on TD	SIG (Nugroho et al., 2011)	(Singh et al., 2014)	(Singh et al., 2016)
Modularity metrics for ATD	-	(Li et al., 2014)	(Li et al., 2014)
Detecting and quantifying SATD	previous version(Potdar and Shihab, 2014)	(Maldonado and Shihab, 2015)	-

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.* (Besker et al., 2016) 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) (Ampatzoglou et al., 2015b) 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.

5 THREATS TO VALIDITY

The main threats to validity identified are the following:

- Although the applied guideline (Kitchenham and Charters, 2007) 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 (August 2018).

6 CONCLUSIONS AND FUTURE WORK

This study provides an overview of the available approaches to the measurement of TD and the tools able to support its automation. The research in the field is very active but there is a lack of validation and evolution of the models. In particular, in almost all cases, models are developed from scratch and not refining or extending existing ones. This shows a very low level of maturity of the field in which it is not clear which is the best approach(s) to follow. Moreover, there is a need of independent validation since nearly none of the models have been independently evaluate but the evaluation is usually performed by the proponents of the approach.

About the tools, they usually require a complex setup, they support a limited number of programming languages, and the results provided are quite different. They also use very different measurement units. Finally, most of the tools are able to estimate only the main TD (sometimes called *remediation cost*), whereas also knowledge of its interest (sometimes called *non-remediation cost*) would complete the picture.

Overall, both methodologies and support tools require a relevant amount of research to become really usable in practice.

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

Technique	**Input**	**Calculation**	**Output**	**Ref**
SQALE	1. Target quality level (a list of non-functional 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)	(Letouzey and Ilkiewicz, 2012)
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	(Curtis et al., 2012)

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
				(Nugroho et al., 2011)

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 (Ploesch et al., 2008) 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	(Mayr et al., 2014)

Table 5: Techniques with input, output, and calculation (cont.).

Technique	**Input**	**Calculation**	**Output**	**Ref**
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	(Skourletopoulos et al., 2015)
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	(Chatzigeorgiou et al., 2015)
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	(Kamei et al., 2016)
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	(Marinescu, 2012)

Table 5: Techniques with input, output, and calculation (cont.).

Technique	**Input**	**Calculation**	**Output**	**Ref**
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	(Singh et al., 2014)
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	(Li et al., 2014)
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)	(Maldonado and Shihab, 2015)

REFERENCES

Scientific toolworks, inc. understand 2.6. http://www.scitools.com/.

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

Ampatzoglou, A., Ampatzoglou, A., Avgeriou, P., and Chatzigeorgiou, A. (2015a). Establishing a framework for managing interest in technical debt. In *5th International Symposium on Business Modeling and Software Design, BMSD*.

Ampatzoglou, A., Ampatzoglou, A., Chatzigeorgiou, A., and Avgeriou, P. (2015b). The financial aspect of managing technical debt: A systematic literature review. *Information and Software Technology*, 64:52–73.

Ayewah, N., Hovemeyer, D., Morgenthaler, J. D., Penix, J., and Pugh, W. (2008). Using static analysis to find bugs. *IEEE software*, 25(5).

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

Besker, T., Martini, A., and Bosch, J. (2016). A systematic literature review and a unified model of atd. In *Software Engineering and Advanced Applications (SEAA), 2016 42th Euromicro Conference on*, pages 189–197. IEEE.

Boehm, B., Grunbacher, P., and Briggs, R. O. (2001). Developing groupware for requirements negotiation: lessons learned. *IEEE software*, 18(3):46–55.

Britsman, E. and Tanriverdi, Ö. (2015). Identifying technical debt impact on maintenance effort-an industrial case study.

Chatzigeorgiou, A., Ampatzoglou, A., Ampatzoglou, A., and Amanatidis, T. (2015). Estimating the breaking point for technical debt. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pages 53–56. IEEE.

Coman, I. D., Robillard, P., Sillitti, A., and Succi, G. (2014). Cooperation, collaboration and pair-programming: Field studies on backup behavior. *Journal of Systems and Software*, 91(5).

Coman, I. D. and Sillitti, A. (2007). An empirical exploratory study on inferring developers' activities from low-level data. In *19th International Conference on Software Engineering and Knowledge Engineering (SEKE 2007)*.

Coman, I. D. and Sillitti, A. (2008). Automated identification of tasks in development sessions. In *16th IEEE International Conference on Program Comprehension (ICPC 2008)*.

Coman, I. D., Sillitti, A., and Succi, G. (2008). 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)*.

Corral, L., Sillitti, A., and Succi, G. (2013). 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*.

Corral, L., Sillitti, A., and Succi, G. (2014). Software assurance practices for mobile applications. *Computing*, 97(10).

Cunningham, W. (1992). The wycash portfolio management system, addendum to the proceedings on object-oriented programming systems, languages, and applications (addendum).

Curtis, B., Sappidi, J., and Szynkarski, A. (2012). Estimating the size, cost, and types of technical debt. In *Proceedings of the Third International Workshop on Managing Technical Debt*, pages 49–53. IEEE Press.

Falessi, D. and Reichel, A. (2015). Towards an open-source tool for measuring and visualizing the interest of technical debt. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pages 1–8. IEEE.

Falessi, D., Shaw, M. A., Shull, F., Mullen, K., and Keymind, M. S. (2013). Practical considerations, challenges, and requirements of tool-support for managing technical debt. In *Managing Technical Debt (MTD), 2013 4th International Workshop on*, pages 16–19. IEEE.

Gaudin, O. (2009). Evaluate your technical debt with sonar. *Sonar, Jun*.

Griffith, I., Reimanis, D., Izurieta, C., Codabux, Z., Deo, A., and Williams, B. (2014). The correspondence between software quality models and technical debt estimation approaches. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, pages 19–26. IEEE.

Gruber, H., Plösch, R., and Saft, M. (2010). On the validity of benchmarking for evaluating code quality. *IWSM/MENSURA*, 10.

Heitlager, I., Kuipers, T., and Visser, J. (2007). A practical model for measuring maintainability. In *Quality of Information and Communications Technology, 2007. QUATIC 2007. 6th International Conference on the*, pages 30–39. IEEE.

Izurieta, C., Griffith, I., Reimanis, D., and Luhr, R. (2013). On the uncertainty of technical debt measurements. In *Information Science and Applications (ICISA), 2013 International Conference on*, pages 1–4. IEEE.

Kamei, Y., Maldonado, E. d. S., Shihab, E., and Ubayashi, N. (2016). Using analytics to quantify interest of self-admitted technical debt. In *QuASoQ/TDA@ APSEC*, pages 68–71.

Kan, S. H. (2002). *Metrics and models in software quality engineering*. Addison-Wesley Longman Publishing Co., Inc.

Kitchenham, B. and Charters, S. (2007). Guidelines for performing systematic literature reviews in software engineering (version 2.3). Technical report, Keele University and University of Durham.

Lenarduzzi, V., Sillitti, A., and Taibi, D. (2017). Analyzing forty years of software maintenance models. In

39th International Conference on Software Engineering (ICSE 2017).

Letouzey, J.-L. (2012). The sqale method for evaluating technical debt. In *Managing Technical Debt (MTD), 2012 Third International Workshop on*, pages 31–36. IEEE.

Letouzey, J.-L. and Ilkiewicz, M. (2012). Managing technical debt with the sqale method. *IEEE software*, 29(6):44–51.

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

Li, Z., Liang, P., Avgeriou, P., Guelfi, N., and Ampatzoglou, A. (2014). 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*, pages 119–128. ACM.

Luhr, R. L. et al. (2015). *The application of technical debt mitigation techniques to a multidisciplinary software project*. PhD thesis, Montana State University-Bozeman, College of Engineering.

Maldonado, E. d. S. and Shihab, E. (2015). Detecting and quantifying different types of self-admitted technical debt. In *Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on*, pages 9–15. IEEE.

Marinescu, R. (2012). Assessing technical debt by identifying design flaws in software systems. *IBM Journal of Research and Development*, 56(5):9–1.

Mayr, A., Plösch, R., and Körner, C. (2014). A benchmarking-based model for technical debt calculation. In *Quality Software (QSIC), 2014 14th International Conference on*, pages 305–314. IEEE.

Monteith, J. Y. and McGregor, J. D. (2013). Exploring software supply chains from a technical debt perspective. In *Proceedings of the 4th International Workshop on Managing Technical Debt*, pages 32–38. IEEE Press.

Moser, R., Pedrycz, W., Sillitti, A., and Succi, G. (2008). A model to identify refactoring effort during maintenance by mining source code repositories. In *9th International Conference on Product Focused Software Process Improvement (PROFES 2008)*.

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

Parodi, E., Matalonga, S., Macchi, D., and Solari, M. (2016). Comparing technical debt in student exercises using test driven development, test last and ad hoc programming. In *Computing Conference (CLEI), 2016 XLII Latin American*, pages 1–10. IEEE.

Ploesch, R., Gruber, H., Pomberger, G., Saft, M., and Schiffer, S. (2008). Tool support for expert-centred code assessments. In *Software Testing, Verification, and Validation, 2008 1st International Conference on*, pages 258–267. IEEE.

Poliakov, D. et al. (2015). A systematic mapping study on technical debt definition.

Potdar, A. and Shihab, E. (2014). An exploratory study on self-admitted technical debt. In *Software Maintenance and Evolution (ICSME), 2014 IEEE International Conference on*, pages 91–100. IEEE.

Ribeiro, L. F., de Freitas Farias, M. A., Mendonça, M. G., and Spínola, R. O. (2016). Decision criteria for the payment of technical debt in software projects: A systematic mapping study. In *ICEIS (1)*, pages 572–579.

Singh, V., Pollock, L. L., Snipes, W., and Kraft, N. A. (2016). A case study of program comprehension effort and technical debt estimations. In *Program Comprehension (ICPC), 2016 IEEE 24th International Conference on*, pages 1–9. IEEE.

Singh, V., Snipes, W., and Kraft, N. A. (2014). A framework for estimating interest on technical debt by monitoring developer activity related to code comprehension. In *Managing Technical Debt (MTD), 2014 Sixth International Workshop on*, pages 27–30. IEEE.

Skourletopoulos, G., Mavromoustakis, C. X., Mastorakis, G., Rodrigues, J. J., Chatzimisios, P., and Batalla, J. M. (2015). A fluctuation-based modelling approach to quantification of the technical debt on mobile cloud-based service level. In *Globecom Workshops (GC Wkshps), 2015 IEEE*, pages 1–6. IEEE.

Snipes, W., Nair, A. R., and Murphy-Hill, E. (2014). Experiences gamifying developer adoption of practices and tools. In *Companion Proceedings of the 36th International Conference on Software Engineering*, pages 105–114. ACM.

Tom, E., Aurum, A., and Vidgen, R. (2013). An exploration of technical debt. *Journal of Systems and Software*, 86(6):1498–1516.

Tsantalis, N., Chaikalis, T., and Chatzigeorgiou, A. (2008). Jdeodorant: Identification and removal of type-checking bad smells. In *Software Maintenance and Reengineering, 2008. CSMR 2008. 12th European Conference on*, pages 329–331. IEEE.

Wheeler, D. A. (2001). More than a gigabuck: Estimating gnu/linux's size.