Monitoring Technical Debt in an Industrial Setting

Elvira-Maria Arvanitou, Apostolos Ampatzoglou, Stamatia Bibi, Alexander Chatzigeorgiou, Ioannis Stamelos

Department of Applied Informatics, University of Macedonia, Thessaloniki, Greece
Computer Science Department, Aristotle University of Thessaloniki, Greece
Department of Informatics and Telecommunications, University of Western Macedonia, Kozani, Greece
earvanitoy@gmail.com, apostolos.ampatzoglou@gmail.com, sbibi@uowm.gr, achat@uom.gr, stamelos@csd.auth.gr

ABSTRACT

Context: Technical Debt (TD) quantification has been studied in the literature and is supported by various tools; however, there is no common ground on what information shall be presented to stakeholders. Similarly to other quality monitoring processes, it is desirable to provide several views of quality through a dashboard, in which metrics concerning the phenomenon of interest are displayed.

Objective: The aim of this study is to investigate the indicators that shall be presented in such a dashboard, so as to: (a) be meaningful for industrial stakeholders, (b) present all necessary information, and (c) be simple enough so that stakeholders can use them.

Method: We explore TD Management (TDM) activities (i.e., measurement, prioritization, repayment) and choose the main concepts that need to be visualized, based on existing literature and tool-support. Next, we perform a survey with 60 software engineers (i.e., architects, developers, etc.) working for 11 software development companies located in 9 countries, to understand their needs for TDM.

Results / Conclusions: The results of the study suggest that different stakeholders need a different view of the quality dashboard, but also some commonalities can be identified. For example, on the one hand, managers are mostly interested in financial concepts, whereas on the other hand developers are more interested in the nature of the problems that exist in the code. The outcomes of this study can be useful to both researchers and practitioners, in the sense that the former can focus their efforts on aspects that are meaningful to industry, whereas the latter to develop meaningful dashboards, with multiple views.

CCS CONCEPTS

Software and its engineering→ Software creation and management→ Software verification and validation→ Empirical software validation

KEYWORDS

Technical debt, visualization, metrics, software quality, survey

EASE '19, April 15–17, 2019, Copenhagen, Denmark

ACM ISBN 978-1-4503-7145-2/19/04…$15.00
https://doi.org/10.1145/3319008.3319019

ACM Reference format:

E. M. Arvanitou, A Ampatzoglou, S. Bibi, A. Chatzigeorgiou, and I. Stamelos, "Monitoring Technical Debt in an Industrial Setting", In *Proceedings of 23rd Conference on the Evaluation and Assessment in Software Engineering (EASE' 19), Copenhagen, Denmark, 15-17April 2019.*

1. Introduction

Technical Debt (TD) is a metaphor from economics that refers to inefficiencies during software development that lead to additional maintenance effort [23]. In recent years, TD has attracted a significant attention from both academia and industry, in the sense that the corpus of academic papers is expanding [1], and the industrial awareness on TD is increasing [4]. The importance of efficient Technical Debt Management (TDM) is highlighted by industrial evidence suggesting that software maintenance, when not performed optimally, can reach up to 75% of the total costs of software development [34]. Additionally, up to 25% of "*wasted*" development time during maintenance can be attributed to TD [29].

According to Li et al. [24], efficient TDM requires the execution of up to eight activities. For simplicity, the activities can be merged to four high-level (HL) activities, based on their goal. We note that only the primitive activities are obtained from Li et al. [24], whereas the synthesized high-level ones are based on our perception of their nature, and conceptual similarities:

- ***Visualizing TD***: The process of visualizing TD includes the activities of: (a) *representing* TD in a uniform manner addressing the concerns of particular stakeholders, (b) *communicating* TD by making it visible to stakeholders so that it can be discussed and further managed, and (c) *monitoring* TD, which deals with observing the evolution of the cost and benefit of unresolved TD over time.
- ***Quantifying TD***: The process of quantifying TD, involves two main activities: (a) *TD identification,* which aims at detecting artifacts that suffer from TD caused by intentional or unintentional technical decisions in a software system through specific techniques, such as static code analysis; and (b) *TD measurement,* which aims at quantifying the benefit and cost of known TD in a software system through estimation techniques, or estimating the level of the overall TD in a system.
- ***Prioritizing TD***: The process of *TD prioritization* ranks identified TD items, according to certain predefined rules to support deciding which TD items should be repaid first and which TD items can be tolerated until later releases.
- ***Reducing TD***: To reduce the amount of TD in a system, two activities can be performed: (a) *TD prevention* that aims to pre-

vent potential TD from being incurred in future developments, and (b) *TD repayment,* which aims to resolve or mitigate TD in a software system by techniques such as reengineering / refactoring.

Given the aforementioned activities, this study focuses on ***Visualizing TD***-related information to support the rest high-level TD activities. More specifically, we focus on the construction of a TDM Dashboard that would present the minimal amount of information that would be required for efficiently supporting the rest of the aforementioned high-level TDM activities, as shown in Figure 1.

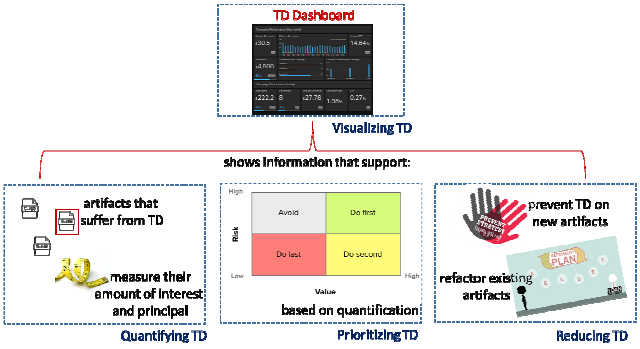

Figure 1. Connection between high-level TDM activities

Visualizing TD is the most understudied activity in TDM, which nevertheless is of paramount importance. The importance of efficient TD visualization is highlighted by the fact that in an industrial context zero TD is an elusive target, which might be considered as not desirable [14]. Additionally, by taking into account the vast amount of information that could potentially be shown in a quality dashboard (i.e., software metrics), it is important to not overload industrial stakeholders with undesirable information. To this end the goal of this study is to extract the needs of industrial stakeholders and determine what to visualize and how to visualize it, in terms of required information for efficient TD quantification, prioritization, and reduction. To achieve this goal, we performed an industrial survey with 60 stakeholders, working for 11 software development companies, located in 9 countries. The rest of the paper is organized as follows: in Section 2 we present related work. In Section 3, we present the candidate pieces of information for inclusion in the TDM dashboard, based on the literature and existing tool-support. In Section 4, we present the survey design, whose results are presented in Section 5. The obtained results are interpreted in Section 6, along with useful implications for researchers and practitioners. Finally, in Section 7, we present threats to validity, and in Section 8 we conclude the paper.

2. Related Work

2.1 Monitoring Technical Debt

According to Li et al. [24], TD representation, communication, and monitoring are among the most understudied TDM activities. Among those studies referenced by Li et al. [24], none is related to the development of a quality dashboard for assisting efficient TDM. Although, tool-wise, state-of-practice platforms, such as SonarQube, or CAST offer TD-related dashboards, these tools are (mostly) focused on TD principal assessment.

From a research point of view, Seaman and Guo [32] review the main issues associated with TD, and propose management mechanisms (processes and tools) for monitoring TD. In particular, they focus on the ongoing monitoring of TD over time, by continuously plotting various aggregated measures over time and look at the shape of the curve to observe the trends. The aggregated measures include: (a) the total number of TD items; (b) the total number of high-principal items; (c) the total number of high interest (probability and amount) items; and (d) the weighted total principal (TP), which is calculated by summing up the items in an entire list (set three points for high, two for medium, one for low TD principal, given some thresholds). This work has set the landscape for TD monitoring, but lacks empirical validation, and information related to activities other than quantification.

Finally, Brown et al. [9] focus on software-reliant systems that have been developed by the agile community. The work focuses mostly on monitoring trade-offs: when developers accept compromises in a system in one dimension (e.g., modularity) to meet an urgent demand in some other dimension (e.g., a deadline). The authors argue that TD needs to be continuously monitored, in the sense that limited TD may not be a problem, which grows when more TD is accumulated. Therefore, there is a need for rules on when TD grows "too much" (e.g., acceptability thresholds) and TD visualization tools. The authors propose the use of a daemon plug-in that demonstrates how to monitor coding rules violations and providing measures using debt heuristics. This plugin should be able to: (a) determine the level of TD over time, (b) recognize trends, and (c) disseminate warnings at appropriate times. Optimally, such tools must be integrated into the IDE.

2.2 Software Quality Dashboards

In this section we discuss research efforts that present the development of software quality dashboards. Software quality dashboards can become extremely relevant when software grows larger through a long evolution history [18]. Heinemann et al. [18] suggest that current quality analysis tools operate in batch-mode and run up to several hours for large systems, which hampers the integration of quality control into daily development. The authors present a quality analysis tool, namely Teamscale, which aims at providing feedback to developers, in limited time at a commit level. The tool has been successfully evaluated within a development team of an insurance company. Additionally, Baysal et al. [8], have performed interviews with Mozilla developers suggesting that there is a need for qualitative dashboards designed to improve developers' awareness by: (a) providing task tracking, (b) presenting insights on the workloads, (c) listing individual issues, etc., to help manage their workloads in day-to-day development tasks. Finally, Steidl et al. [33], suggest that although companies often use static analysis tools, they do not derive consequences from the metric results and, hence, the code quality does not actually improve. The authors present an experience report of a consulting company, suggesting how code quality can be improved in practice, by combining metrics, manual action, requiring however the close cooperation between quality managers, developers, and managers.

2.3 Perception of TD in Industry

In this section we discuss similar (in terms of research method) studies, which have surveyed industrial stakeholders to mainly understand their perception on TD. Lim et al. [28] performed an interview study, which aimed to characterize TD at the ground level and understand the context in which it occurs and how software practitioners

perceive it. The main outcomes of the study are: (a) developers are familiar with TD, since they "*have to live with it everyday*"; and (b) measuring TD is not an easy task, because its impact is not uniform. Additionally, Ernst et al. [15], report the results of a survey with 1,831 participants, primarily software engineers and architects working in long-lived, software-intensive projects from three large organizations, and follow-up interviews of seven software engineers. The findings suggested that architectural decisions are the most important source of TD. Furthermore, while respondents believe the metaphor is itself important for communication, existing tools are not currently helpful in managing the details. Their results are used to motivate a technical debt timeline to focus management and tooling approaches. Finally, Ampatzoglou et al. [4] conducted a supervised survey in the embedded systems industry, to investigate: (a) the expected life-time of components that have TD, (b) the most frequently occurring types of TD in them, and (c) the significance of TD against run-time quality attributes. The results suggested that: (a) maintainability is more seriously considered when the expected lifetime of components is larger than ten years, (b) the most frequent types of debt are test, architectural, and code debt, and (c) in embedded systems the run-time qualities are prioritized compared to design-time ones (associated with TD). Finally, Martini and Bosch [25] performed a survey with TD practitioners to identify the motivation for performing TD prioritization and refactoring. The results suggested that although competitive advantage and attractiveness are very important aspects, almost no ATD approach uses them for prioritization.

3. Dashboard Views & Information Alternatives

In this section we present the envisioned views of the TDM dashboard and the tentative pieces of information that it could visualize for supporting TDM activities. The selection of views is driven by the TDM activities (see Section 1), whereas the alternative types of information have been retrieved based on existing literature and available tools.

3.1 TD Quantification Dashboard

The cornerstones of the TD metaphor are two concepts borrowed from economics: namely principal and interest. On the one hand, ***TD principa*l** is the effort required to eliminate inefficiencies in the current design or implementation of a software system [3]. On the contrary, ***TD interest*** is the additional development effort required to modify the software, due to the presence of said inefficiencies. For instance, when new functionality needs to be added, additional effort needs to be spent due to inferior design quality [10].

According to two recent secondary studies on TD management by Ampatzoglou et al. [3] and Li et al. [24], SonarQube is the most frequently used tool for estimating ***TD principal***. SonarQube is representing TD principal through two different views: (a) the number of inefficiencies in the source code, and (b) the amount of time required to fix such inefficiencies. However, since the TD concept relies on monetary assessments a third option has been considered: (c) the monetary assessment of the aforementioned effort in US dollars/euros. Apart from these system-wide views SonarQube provides the opportunity to focus on specific artifacts that suffer from TD, and also provides a listing of the kind of problems that are identified.

Software maintainability is inherently related to technical debt, and in particular to ***TD interest*** [23] (i.e., how easy it is for a software engineer to apply changes in a specific software system). So, we consider maintainability as a proxy for TD. Maintainability, although not associated to a universally accepted definition, is widely perceived as the ease of making changes into a system. Moreover, even when a practitioner is not aware of the incurred TD, any decision that aims at enhancing maintainability will result in lowering the amount of TD interest. On the other hand, if a development team is not interested in producing a maintainable system, then it is highly probable that shortcuts will be made; subsequently these shortcuts will hinder any future maintenance activity. In the literature one can identify various models for assessing maintainability, through corresponding tools.

3.2 TD Prioritization Dashboard

According to Galorath [17] and Chen and Huang [12] maintenance costs are increased by up to 75% if the software is unstable. In the literature, one can identify a relevant quality property, termed stability (and it's opposite: instability) [19]. Based on the ISO-9126 standard *stability characterizes the sensitivity of a given system to change, which is the negative impact that may be caused by system changes* [19]. In the technical debt literature, instability is considered as a proxy of ***interest probability***. More specifically, it is claimed that more change-prone artifacts are more likely to accumulate interest than less change-prone ones, since interest manifests only during maintenance activities [5]. Additionally, a class that will never change along evolution, regardless of how poorly designed it is (i.e., high principal) will never produce interest if it is not maintained. According to Seaman and Guo [32], TD prioritization can be performed, either based on principal, interest, or interest probability. Given the fact that principal and interest information have been presented in the previous view, the TD prioritization view shall focus on interest probability.

3.3 TD Reduction Strategy Dashboard

The amount of TD in a system can be mitigated in two ways: (a) repay existing TD by applying refactoring, or (b) prevent the accumulation of TD, by writing new clean code. The TD reduction dashboard can potentially present information for both strategies. On the one hand, regarding the ***"cleanness" of new code***, the dashboard could present a comparison between the TD intensity (i.e., the fraction of the total TD divided by the Lines of Code of the artifact under study) in existing against the TD intensity on new code [13]. The rationale for focusing on cleaner new code, is that by committing code that is of superior quality than the existing average, the entire codebase will eventually improve in terms of TD.

On the other hand, regarding the ***refactoring of existing code***, in the literature one can identify various indicators or refactoring opportunities identification tools (e.g., [6][26]). Such methods and tools can vary across various levels of granularity, ranging from the source code level to the architecture level. For instance, with respect to the violation of the Single Responsibility Principle, we have been able to identify tools that can perform extract method opportunities (e.g., [11]), i.e., working at the micro-level, whereas other are able to split long packages (e.g., [31]), working at the architecture level.

4. Study Design

According to Pfleeger and Kitchenham [30], surveys are the most appropriate research method for gathering information to describe existing knowledge, attitudes, or behavior. Surveys are used to gather

information on topics with which the subjects are familiar. For the case of this study, although subjects might not be extremely familiar with the TD concepts and terminology, all subjects are experienced in issues related to quality assessment. A possible lack of experience in TD terminology has been considered during the design of the data collection instrument (see Section 4.2).

The survey is organized based on the activities defined by Pfleeger and Kitchenham [30]: (a) set research objectives, (b) plan and schedule the survey, (c) ensure that appropriate resources are available, (d) design the survey, (e) prepare the data collection instrument, (f) validate the instrument, (g) select participants, (h) administer and score the instrument, (i) analyze data, and (j) report the results. To avoid excessive use of sub-sectioning, we present activities (a–d and g) in Section 4.1 (namely "*Survey Design"*), activities (e, f and h) in Section 4.2 (namely "*Survey Instrument Design"*), activity (i) in Section 4.3 (namely "*Data Analysis Strategy"*) and activity (j) in Section 5 (namely "*Results"*).

4.1 Survey Design

The survey design section presents research objectives and research questions, survey planning, resource management and selection of participants. The design process began with reviewing the objectives, examining the target population identified by the objectives and deciding how the data collection shall be approached for obtaining the information needed to address those objectives. Additionally, we considered factors such as: (a) determining the appropriate sample size, and (b) ensuring the largest possible response rate [30].

Research Objective: The goal of this survey, formulated as a GQM statement [7], is to: "*analyze* industrial stakeholders' concerns *for the purpose of* understanding their needs *with respect to* the information that should be visualized in a TDM dashboard for supporting: (a) TD Quantification, (b) TD Prioritization, and (c) TD Reduction *from the point of view* of the various roles of the stakeholders".

Research Questions: Based on the aforementioned goal we were able to state four research questions that will guide the design of this survey and the reporting of the results:

RQ_1: *What could be the optimal information to be visualized in a TDM dashboard for efficient TD quantification?*

RQ_1 aims to explore which are the most important indicators that one stakeholder would consider beneficial for TD monitoring. In particular, we provide indicators for TD principal and TD interest (see Section 3). The motivation for setting up this research question is that when dealing with stakeholders with very limited time "*less is more*". In other words, it is expected that an optimal dashboard shall provide to practitioners, exactly the amount of information that they need, without any extra material that would not be useful and would only hinder the quality assessment process. Such an approach is expected to increase the usability of the dashboard, the satisfaction of the involved stakeholders, and therefore the efficiency of technical debt management.

RQ_2: *What could be the optimal information to be visualized in a TDM dashboard for efficient TD prioritization?*

RQ_2 deals with the information required for efficient prioritization of individual inefficiencies. According to Seaman and Guo [32], there are three ways that one can prioritize which TD items (TDIs—i.e., artifacts that suffer from TD) to refactor first those with the highest: (a) principal, (b) interest, or (c) interest probability. By considering that (a) and (b) have already been explored in RQ_1, in this research question we focus on interest probability (c) and in particular in its perceived usefulness, and the extent to which stakeholders would be interested to include it in a TDM dashboard. Additionally, we investigate if the kind of the identified inefficiency can influence TDI prioritization. Additionally, we dig further in this task by investigating the level of granularity (e.g., method- or architecture-level) that seems more appealing to stakeholders, regarding TD prioritization. On the one hand, changes at the higher levels of granularity are expected to have a larger impact on quality, but on the other hand, such changes might seem a bit abstract to practitioners and thereof difficult to apply.

RQ_3: *What could be the optimal information to be visualized in a TDM dashboard for efficient TD reduction?*

In the literature, one can identify two distinct ways to reduce the normalized amount of TD that is accumulated in a software system: (a) preventing the accumulation of TD in new artifacts, and (b) repaying the TD that is already accumulated into existing code. In RQ_3, we contrast these two options, by weighting the importance of incorporating them in the TDM dashboard. The normalized amount of TD is considered important so as to be able to compare TD accumulation of systems with different (however, not substantially different) sizes.

RQ_4: *Do different roles of stakeholders require different views of the quality dashboard?*

RQ_4 deals with investigating if the different roles of stakeholders, drives the incorporation of multiple views in the TDM dashboard. For example, we expect that managers will be more interested on monetary views of software quality (since their responsibility is to get an overview of system quality and assess the required effort), compared to software developers, who would be more interested in getting indicators on the artifacts that are suffering from TD (since it is their responsibility to resolve them).

Design: The goal of this survey is to identify the needs of industrial stakeholders, for the purpose of developing a dashboard that would include all necessary (but minimal) information for efficient TDM. Based on the nature and the special characteristics of survey design, it has been organized as a not supervised, cross sectional study [20]. The study is not supervised, because the researchers have not intervened while participants fill in the survey instrument. In addition, it is cross-sectional, because participants have been asked about their past experiences in a given point in time [20].

Plan and Schedule: According to Kitchenham and Pfleeger, there are six common ways to get information: literature searches, personal interviews, focus groups, email or telephone surveys, and online questionnaires [20]. In this survey, we performed data collection through an online questionnaire, so as to increase the number of possible participants, since the supervision will not be necessary. To increase the response rate, we sent invitation emails in two phases (an initial one, and a reminder). The reminder has been sent two weeks after the original email, and we stopped waiting for answers, one month after sending the reminder. The survey was executed between October 2018 and January 2019: from October 2018 till mid-November the survey was designed and piloted, data extraction lasted until mid-December, whereas data analysis and reporting was performed from mid-December to mid-January.

Resource Management: Online surveys are the most cost-effective method of distributing a survey. The use of Google Forms provides us the opportunity to easily setup the survey instrument and distribute it, among practitioners, whereas all responses were automatically managed by Google. The main benefit of this strategy is that no errors during the recording of the responses can be introduced.

Participants Selection: As participants we opted for stakeholders with different roles in the software industry. This survey has been conducted as part of two research projects, namely: SDK4ED and EXTREME (see Acknowledgements), and therefore participants have been retrieved from industrial partners of the two consortia, which are spread across EU, and vary in terms of application domains. For confidentiality reasons the companies have been anonymized in this manuscript. The publicly available information is presented in Table I.

TABLE I. PARTICIPATING COMPANIES DEMOGRAPHICS

ID	Application Domain	Country	Participants	Size
C1	Airborne	France	2	Large
C2	Constructions	Greece	8	Large
C3	Embedded Systems	Sweden	3	SME
C4	Quality Assurance	Germany	6	SME
C5	Transportation	Belgium	2	SME
C6	Augmented Reality	Romania	3	SME
C7	Mobile Applications	Greece	5	SME
C8	Enterprise Applications	Luxemburg	13	Large
C9	Enterprise Applications	Greece	14	Large
C10	Medical Applications	Netherlands	1	SME
C11	Mobile Application	Cyprus	3	SME

To reach the most fitting participants for our study, we have not blindly targeted all software engineers of the eleven involved companies, but we have only reached project managers and asked them to forward the questionnaire to people that would be candidate stakeholders of the obtained dashboard and are experienced in such tasks. From SMEs we asked them to forward the email to 10 individuals, and from Large Enterprises to forward the email to 20 people. Based on our planning 150 invitations have been sent. The response rate that we have achieved was 40%, which is substantially higher than the expected one (according to Kitchenham and Pfleeger [20] 20% is an acceptable response rate). This high response rate can be attributed to two possible reasons: (a) the industries are already collaborating ones, so an established connection of trust has been used, and (b) the selection of participants from the project managers was beneficial, since they were aware of subjects that would be interested in the survey scope and goals. To comply with GDPR, we informed the participants of the survey that: (a) the results of the study will be made available to them in an aggregate form in case they are interested; (b) will only be published in an aggregated form; (c) each participant should proceed with the completion of the questionnaire only if he/she provides his consent, and (d) their data will be erased upon participants requests.

4.2 Survey Instrument Design

Survey instruments are questionnaires that are constructed in three steps: (a) preparation, (b) evaluation, and (c) documentation [21].

Prepare the Data Collection Instrument: The most important part of developing a questionnaire is the selection of questions. In our study, this process was governed by the guidelines provided in [21]: (a) keep the amount of questions low, (b) questions should be purposeful and concrete, (c) answer categories should be mutually exclusive, and (d) they should avoid biasing the respondent.

To this end, we constructed a questionnaire with 15 main questions, organized into three main sections (see Table II), and an introductory one (2 questions). The questionnaire begins with some demographic information (Name of Company and Role in the Company). We note that we have not prompted the participants to record their experience, since they were all considered as experienced enough from their managers, who invited them to participate in the survey. In Section-2, industrial stakeholders are asked to rate, in a Likert scale, a group of questions based on the usefulness of TD principal indicators, whereas in Section-3, he/she is asked to rate, a group of questions based on the usefulness of TD interest indicators. Additionally, in Section-4, he/she is asked to consider the optimal strategy for mitigating TD. In the beginning of each Section some basic TD definitions have been provided, so as to establish a common understanding and terminology among participants. The complete questionnaire is available online. The mapping of questions to RQs is provided in Section 4.3.

TABLE II. SURVEY INSTRUMENT

ID	Question
Section 2 – TD Principal	
Q.2.1	How useful is it to know how many inefficiencies your code has?
Q.2.2	How useful is it to have an estimation of the effort required to solve all inefficiencies?
Q.2.3	How useful is it to have an estimation of the effort required to solve all inefficiencies in a currency format (e.g. $ or €)?
Q.2.4	If you had only one option for quantifying inefficiencies, which one would you prefer?
Q.2.5	To what extent the granularity (architecture, method) of an inefficiency is important for selecting the ones to resolve?
Section 3 – TD Interest / Interest Probability	
Q.3.1	How useful is it to know how maintainable your code as a whole (system-level) is?
Q.3.2	How useful is it to know the maintainability of specific artifacts?
Q.3.3	How useful is it to know the kind of structural problems that hinder maintainability?
Q.3.4	How useful is it to know the probability of one artifact to need maintenance?
Q.3.5	At which level of granularity would you prefer to get a maintainability indicator?
Q.3.6	What piece of information would be more important for selecting which artifact to refactor first?
Section 4 – TD Reduction	
Q.4.1	Do you consider TD repayment (refactoring) as a useful activity for improving quality?
Q.4.2	Do you consider writing new code that is TD-free as a useful activity for improving quality?
Q.4.3	Do you consider a metric comparing TD density on existing and new code useful?
Q.4.4	To reduce the TD density of your system, would you prefer to refactor or write TD-free new code?

The majority of the questions have been answered in a Likert Scale ranging from: (a) "*Not Useful*" to "*Very Useful*" for questions that

start with "*How useful...?*", or *"Do you consider…"*, and (b) "*Not Important"* to "*Very Important"* for the question Q.2.5. Questions that do not fall in the aforementioned three categories, the candidate responses were as follows:

2.4 (a) number of inefficiencies, (b) effort to solve inefficiencies, or (c) monetary assessment of the effort required to solve inefficiencies

3.5 (a) system-level maintainability, or (b) artifact-level maintainability

3.6 (a) kind of maintenance problems, or (b) probability of an artifact to need maintenance

4.4 (a) refactoring, or (b) write TD-free new code

Evaluation: Before data collection, the survey instrument should be evaluated [21]. In particular, we performed a pilot survey with a smaller number of participants (i.e., PhD and MSc students with substantial software engineering experience), to check the understandability of the questions, the reliability and validity of the survey instrument, and the appropriateness of the data analysis techniques [21]. Especially for ensuring the validity of the process (by testing the consistency of respondents' answers), we have inserted some control questions (see Section 4.3). For example, with respect to the quantification of TD principal questions Q.2.1 – Q.2.3 rate alternative forms of representation individually. Next, in Q.2.4 (the control question), we ask participants to select one of the three. We expect participants in Q.2.4 to select the indicator with the max score of Q.2.1 – Q.2.3.We note that the list of questions that was listed in Table II is the final one, after the execution of the pilot, which led to mostly syntax changes.

Documentation: Since our survey was self-administered, we have developed a questionnaire specification (survey protocol) including: (a) the objectives of the study, (b) the description of the rationale for each question, and (c) the description of the evaluation process. In the end, when the questionnaire is administered, we updated the questionnaire specification with more information [21].

4.3 Data Analysis Strategy

Our dataset consists of 16 columns (questions in Table II, plus the role of the respondent in the company) and 60 rows (responses). The obtained dataset consists of 8 managers, 12 architects/designers, 3 requirements' engineers, 8 testers, and 29 software developers. The data analysis [22] has three goals: (a) evaluate the correctness of the developed questionnaire, (b) identify the most useful information to be used in a TDM dashboard (by answering the RQs of the study), and (c) study potential differences the views of different stakeholder roles.

To achieve ***goal (a)***, we have used four control questions: Q.2.4 Q.3.5, Q.3.6, and Q.4.4. The control questions are marked with italic fonts inside a parenthesis in Table III—the parenthesis suggests the questions that are being tested. To evaluate the correctness of the questionnaire, we have examined the consistency of the obtained answers similarly to the evaluation of the survey instrument, as exemplified in Section 4.2. The reporting of consistency will be performed through frequencies: i.e., the percentage of responses to control questions that are in accordance to the responses to the individual answers to which they are mapped to (e.g., Q.2.4 is mapped to Q.2.1 – Q.2.3).

To achieve ***goal (b)***, we have used descriptive statistical analysis on the obtained answers. The mapping of questionnaire questions and research questions is presented in Table III. Analysis with frequency tables will reveal the importance of the parameter studied in each question. For visualization, pie and bar charts will be created. We note that we use pie and bar charts as means of visualization, instead of boxplots, because our variables are ordinal, and therefore treating them as numeric values would not be appropriate [16]. For similar reasons, we have selected not to perform paired sample t-tests, for comparing means or use 95% confidence intervals.

TABLE III. ANALYSIS STRATEGY

RQ	Questions
1	Q.2.1 – Q.2.3 (*Q.2.4*), Q.3.1, Q.3.2 (*Q3.5*)
2	Q.3.3 – Q.3.4 (*Q.3.6*), Q.2.5
3	Q.4.1 – Q.4.3 (*Q.4.4*)
4	All Questions + Role

To achieve ***goal (c)***, we have performed cross tabulation between stakeholder roles and frequencies of answers to each question of the survey instrument, and calculated the chi-square index to check if there are differences among the answers of stakeholders of various roles. The mapping of questions to RQs is the same as in goal (b)—see Table III. Similarly to goal (b), we have not been able to perform Analysis of Variance, since the variables are ordinal.

5. Results

In this section we present the results of this study, organized based on the goals that have been described in Section 4.3. In Section 5.1 we present the instrument validation, in Section 5.2 the answers to RQ_1-RQ_3 (TDM dashboard), and in Section 5.3 we present the differences in the perception of stakeholders with different roles (RQ_4).

5.1 Instrument Validity

The results of instrument validation suggest that the questionnaire is well-constructed regarding the internal consistency of the instrument. The results are summarized in Table IV. The rows of the table correspond to sets of control and individual questions, whereas the last column presents the levels of consistency.

TABLE IV. INSTRUMENT VALIDATION

Control Question	Individual Questions	Consistency
Q.2.4	Q.2.1 – Q.2.3	81%
Q.3.5	Q.3.1 – Q.3.2	91%
Q.3.6	Q.3.3 – Q.3.4	96%
Q.4.4	Q.4.1 – Q.4.2	84%

For example, the 2nd row suggests that 91% of the respondents gave consistent answers to Q.3.1, Q.3.2, and Q.3.5. For this row as consistent we consider the following cases:

- ***stakeholders*** that have selected ***"system-level maintainability" in Q.3.5*** have rated Q.3.1 higher or equal than Q.3.2
- ***stakeholders*** that have selected ***"artifact-level maintainability" in Q.3.5*** have rated Q.3.2 higher or equal than Q.3.1

The survey instrument is considered as internally consistent at the 80% level, which is interpreted as high internal consistency.

5.2 TDM Dashboard Setup

TD Quantification Dashboard: To investigate the indicators that are preferable for the quantification of TD (RQ_1), we have investigated

the indicators for measuring the two main concepts of the TD metaphor: principal and interest. The results of the survey (Q.2.4—see Figure 2) suggest that an indication of the "*effort required to resolve all inefficiencies in minutes"* seems to be the most acceptable view of TD principal, followed by the "*number of inefficiencies*", and the "*effort required to resolve all inefficiencies in currency*". The frequency analysis of individual questions (Q.2.1-Q.2.3) yielded similar results in the sense that the mode and median value for "*effort in minutes"* and "*number of inefficiencies"* was "Very Useful", whereas for "*effort in currency*" the median value was "Useful" (the mode value is "Very Useful").

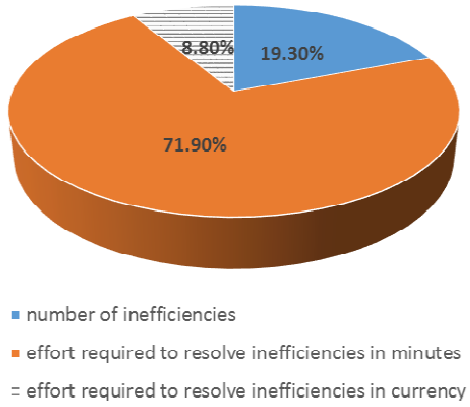

Figure 2. TD Principal Indicators

Regarding the level of granularity at which TD interest shall be calculated, the results of the study suggest that an assessment and the reporting of the TD interest at the artifact-level is more useful compared to the system-level. In Figure 3, we present the fraction of stakeholders that selected each option in the corresponding control question (Q.3.5). The results suggest that more than 50% of the respondents are interested in artifact-based assessments. However, in addition to the close difference presented in Figure 3, the investigation of the responses of stakeholders to individual questions (Q.3.1-Q.3.2), confirmed that this result is rather marginal (see Figure 4), and therefore both options have a merit of their own.

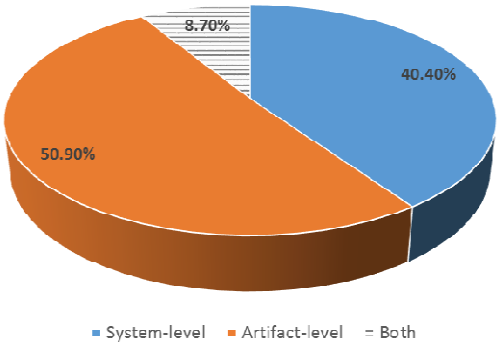

Figure 3. Level of Granularity for Calculating TD Interest

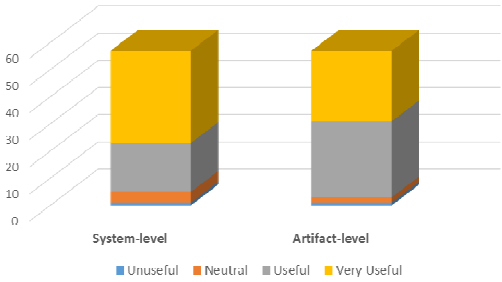

Figure 4. TD Interest Preferences (stacked bars)

TD quantification is preferable at the artifact level, however, system-level assessments are also deemed important, and they shall not be disregarded. Indicators that offer an assessment of TD concepts in an effort form (in minutes) are preferable compared to a simple count of inefficiencies.

TD Prioritization Dashboard: To unveil the information that is necessary for TDIs prioritization, we have explored two possible parameters: (a) the artifact to change—and in particular its change probability, and (b) the kind of the problem from which the artifact suffers from. The results (control question Q.3.6) suggest that the kind of problem from which the artifact suffers from is more important for prioritization purposes, compared to the artifact per se (kind of problem: 57%, specific artifact: 37%, both: 6%). An outline of the stakeholders' answers to individual questions (Q.3.3 and Q.3.4) is provided in Figure 5. The frequency analysis suggests that the mode and median value on the usefulness of the kind of problem information is "*Very Useful*", whereas for the usefulness of knowing the information "which artifact is going to change" is "*Useful*".

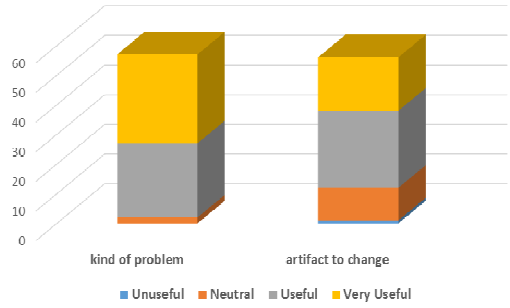

Figure 5. TD Prioritization (stacked bars)

Similarly to before, the importance of knowing the level of granularity of the identified inefficiency for prioritizing the resolution of the TDI that is suffering from it is being explored (Q.2.5). The results suggest that the level of granularity of the inefficiency (i.e., method, class, or architecture level) is an important parameter for the repayment decision—mode and median value: "*Very Important*".

TD prioritization is preferable to be performed on artifacts that suffer from certain design problems (the level of granularity is an important parameter), rather than on artifacts that are changing frequently.

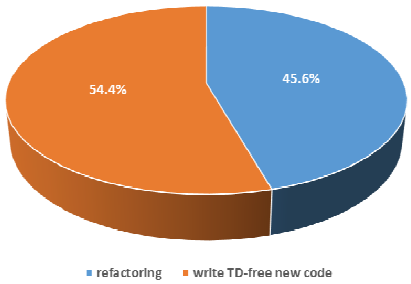

Figure 6. TD Reduction Preferences (pie-chart)

TD Reduction Dashboard: By comparing the preferable strategy of stakeholders to reduce the amount of technical debt, our results (Q.4.4) suggest that writing clean code on new artifacts is preferable compared to refactoring, see Figure 6. The mode and median value for Q.4.1 and Q.4.2 are both "Very Useful". This belief is supported by the fact that stakeholders consider the existence of a metric that capture the amount of TD inserted with new code as (at least) useful by 75% (Q.4.3).

Stakeholders prefer to reduce TD by writing TD-free new code on new modules, compared to refactoring existing artifacts.

5.3 Roles of Stakeholders

Regarding the different perception that the role of each stakeholder can bring into the TDM dashboard configuration, we can observe that important differences can be identified only in the quantification of TD principal, followed by the level of reporting interest, and the role of granularity in TD prioritization. In particular, in Table V, we present the results of the chi-square test after cross-tabulating the role of the stakeholders and the answers obtained from control questions in the survey instrument.

TABLE V. IMPACT OF STAKEHOLDERS' VIEW

RQ	Q	Description	x^2	sig.
1	2.4	TD Principal Quantification	13.81	0.02
	3.1	Interest at System-Level	7.39	0.83
	3.2	Interest at Artifact-Level	17.03	0.04
2	3.3	Prioritze based on Kind of Problem	4.74	0.78
	3.4	Prioritze based on Artifact's Interest Probability	10.19	0.59
	2.5	Prioritize based on Problem Granularity	9.88	0.05
3	4.4	Repayment Strategy (refactor vs. new code)	4.58	0.33

From the aforementioned results, we can observe that only three are statistically significant at the 0.05 level (Q.2.4, Q.3.2, and Q.2.5) and therefore warrant additional investigation. Therefore, for each of these questions, we present a heatmap (see Figures 7a-7c). In the heatmap, the rows correspond to the role of the stakeholder, and the columns to the tentative answers. The color of each cell suggests if the value for the specific type is higher (green) or lower (red) than the expected one. The percentage that appears inside the cell corresponds to the fraction (expressed as a percentage) of the difference between the observed value in the Likert scale from the expected value, over the expected value.

	effort in currency	effort in minutes	number of inefficiencies
Manager	389%	-40%	-26%
Requirements Engineering	-100%	39%	-100%
Architect-Designer	-100%	4%	30%
Software Developer	-43%	1%	17%
Tester-Quality Assurance	-100%	39%	-100%

Figure 7a. TD Principal Quantification

	Not Useful	Neutral	Useful	Very Useful
Manager	-100%	307%	45%	-69%
Requirements Engineering	-100%	-100%	104%	-100%
Architect-Designer	-100%	-100%	2%	10%
Software Developer	-100%	-29%	-8%	15%
Tester-Quality Assurance	1040%	-100%	-19%	-12%

Figure 7b. Interest Reporting at Artifact Level

	Neutral	Useful	Very Useful
Manager	81%	48%	-69%
Requirements Engineering	533%	-100%	-100%
Architect-Designer	-100%	30%	10%
Software Developer	-21%	-9%	15%
Tester-Quality Assurance	27%	4%	-12%

Figure 7c. Prioritization based on Problem Granularity

The role of stakeholders does not appear to substantially influence the views that need to be included in the TDM dashboard. However, some deviations have been identified: (a) managers are more interested in monetary views of problems, (b) testers and managers are those that are least interested in quality at artifact-level, since they prefer having the big picture, and (c) requirements engineers do not see any benefit of refactoring inefficiencies based on their level of granularity.

6. Discussion

In this section we discuss the findings of this study, organized into two sub-sections: in Section 6.1 we provide tentative interpretations of the obtained results, whereas is Section 6.2 we discuss the implications to research and practice from this study.

6.1 Interpretations of Results

The main findings of this work are summarized and discussed below:

- ***TD quantification***: The results suggest that stakeholders prefer to see effort-related information of inefficiencies and improperly designed/developed artifacts. This preference suggests that the TD metaphor (relating poor software development to effort is useful in practice. The fact that the currency view is not the most popular among stakeholders, can be interpreted by the fact that most industrial stakeholders are more familiar with effort in MMs or minutes, while monetary estimates are by definition subject to numerous assumptions.

- ***Level of Detail for Reporting***: The results suggest that a marginal majority of stakeholders suggest that the level of reporting shall be at the artifact level, and not system-wide. This finding is interpreted by the fact that artifact-level reporting can be more easily related to actionable results in the sense that it points to part of the system that need redesign. On the contrary, system-wide reporting provides a panoramic view of quality, which does not lead to specific actions. However, such an assessment still has a merit for comparing different software systems, as a whole.
- ***TD Prioritization parameters***: Stakeholders believe that they shall start repaying TD from artifacts that suffer from specific problems, and not based on the interest probability of the artifact. This outcome, although not intuitive is supported by the literature [2], and can be partially explained due to the fact that stakeholders are not acquainted with TD terminology and principles. Thus, it might not be clear to them that additional parameters shall be considered.
- ***TD Repayment strategies***: Writing TD-free new code, or developing TD-free new artifacts seems as a more promising strategy for mitigating TD, compared to refactoring existing code. This finding suggests that stakeholders are reluctant to change a piece of software that is well-tested and properly running, just for the sake of improving quality. Therefore, they find the 'clean new code' approach as more intriguing, in the sense that minimal investment during commits can ensure software with less TD on average.
- ***Managers' View***: The main differences of the managers' view compared to the study corpus are that they are more interested: (a) in monetary views and (b) system-wide evaluations, compared to the rest stakeholders. The results can be considered intuitive, since managers are more familiar to working with monetary estimates, and they are not interested in details. The first observation confirms the belief of the TD community that the TD metaphor can bridge the gap between technical and managerial stakeholders.
- ***Testers' View***: Testers are among the least interested in artifact-level reporting. However, this result shall be treated with caution, since only eight testers exist in our sample. Although initially this result seems as counter-intuitive, it can be explained if the testers' role in the company is the reporting of quality at a more coarse-grain level through quality gates, which are either passed or failed.

6.2 Implications to Researchers and Practitioners

The results of this study lead to interesting implications for both researchers and practitioners. Regarding researchers we highlight that:

- The TD community shall try to further ***disseminate basic views of the TD metaphor, e.g., interest probability***. At this point, although industrial stakeholders clearly understand the concepts of principal and interest, they are neglecting the important parameter of interest probability. This parameter is acknowledged as important in TD research cycles (e.g., [5], [32]), but it still is not acknowledged as important by the practitioners.
- ***Investigate the benefits of developing TD-free artifacts***. This approach seems promising to practitioners, but there exists limited empirical evidence that such a strategy will have the same effect as refactoring of a system. Therefore, there is a need for rigorously investigating an industrial relevant topic.
- ***TD as a vehicle for communication***. This study provides some initial empirical evidence that TD can play the role of a communication vehicle among technical and managerial stakeholders. However, this belief needs to be further supported by strong empirical evidence.

Regarding practitioners, the outcomes of this study are able to drive the development of an industrially-relevant TDM platform that covers the needs of stakeholders and eventually lead to efficient TDM. Some initial directions on the features that such a TDM dashboard could have are discussed next. Figure 7 presents a mockup of a TD quantification and evolution dashboard. First, in the left side of the screen, the user sees the project explorer tree, through which he/she can select the artifact (either the class or the package) that he/she wants to check. In the TD principal panel, an overview of the results (similar to those obtained from the SonarQube tool [27]) is presented (e.g., cumulative TD principal, number of bugs, violations, code smells etc.). In the TD Interest panel, one can inspect the total interest, the levels of maintainability predictors, the measurements of change proneness (interest probability, instability and interest probability ranking), and the table that contains the top-10 probable inefficiencies. In the TD of New Code the evolution of technical debt for the newly added code vs. the technical debt of existing code will be presented.

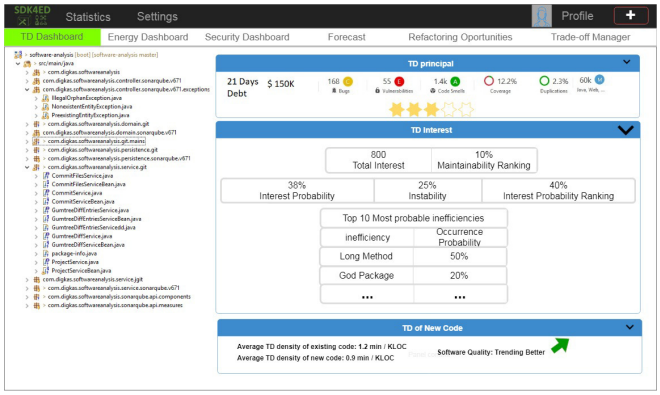

Figure 7. TDM Dashboard Overview

TD principal indicators are expressed in effort related-information (both currency and minutes, so as to cover all stakeholders). Furthermore, both interest and principal indicators can be easily swap from system-level to artifact-level using the tree-view. Additionally, TD on new code has a prominent place in the dashboard. Interest probability information is presented, along with information on the types of violations. The developed demo dashboards are part of SDK4ED platform, and more details can be found online.

7. Threats to Validity

While designing this study, we have identified several threats to validity. First, regarding conclusion validity, all interpretations are tentative ones, since (by definition) surveys cannot support causality, but only report trends and general beliefs in the state-of-practice. Additionally, the sample of this study is a bit narrow compared to other surveys; however, it could not be expanded to a larger population since we were interested in a corpus of software engineers that are experienced and are aware of software quality assessment practices. Nevertheless, we need to note that the wide-spread of the population to many companies, that vary across EU countries guarantee to some extent the generalizability of the results. Furthermore, we acknowledge that repeating the study with a different set of industrial stakeholders might

yield different results; however, the study design is completely replicable since all data collection instruments and procedures are presented transparently in Section 4. Finally, a threat to construct validity stems from the fact that we presented to the participants only elements retrieved from the literature or existing tools; therefore, we might have missed other aspects that they consider important, but were not listed in tentative answers, neither including open-ended questions.

8. Conclusions

TD is a powerful metaphor that has been recently used to raise the awareness of software developers to quality malfunctions. TD is a continuously evolving concept that needs monitoring and corresponding managing activities. However, neither in state-of-research or –practice one can identify guidelines on how to visualize TD, through a simple, but holistic dashboard. In this paper, we aim to obtain the industrial requirements for such a dashboard through a survey with experienced software engineers. The results of the study suggest that TD metaphor concepts, such as principle and interest, make sense to the practitioners, since the effort-related estimates that they provide appear to be useful. Additionally, several parameters for TD prioritization and repayment have been investigated and useful conclusions have been reached. As a final step to this study, we drafted a mockup of the envisioned dashboard that we are currently implementing as part of the SDK4ED project.

ACKNOWLEDGEMENTS

Work reported in this paper has received funding from the European Union Horizon 2020 research and innovation programme under grant agreement No. 780572 (project: SDK4ED).

REFERENCES

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

[2] T. Amanatidis, N. Mittas, A. Chatzigeorgiou, A. Ampatzoglou, and L. Angelis, "The developer's dilemma: factors affecting the decision to repay code debt", *Proceedings of the 2018 International Conference on Technical Debt (TechDebt' 18)*, IEEE, pp. 62-66, Gothenburg, Sweden, June 2018.

[3] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review", Information and Software Technology, Elsevier, vol. 64, pp. 52–73, Aug. 2015.

[4] Ar. Ampatzoglou, Ap. Ampatzoglou, A. Chatzigeorgiou, P. Avgeriou, P. Abrahamsson, A. Martini, U. Zdun and K. Systa, "The Perception of Technical Debt in the Embedded Systems Domain: An Industrial Case Study", 8th International Workshop on Managing Technical Debt (MTD' 16), IEEE, USA, Oct. 2016.

[5] Ar. Ampatzoglou, Ap. Ampatzoglou, P. Avgeriou, and A. Chatzigeorgiou, "Establishing a framework for managing interest in technical debt", 5th International Symposium on Business Modeling and Software Design (BMSD'15), July 2015.

[6] F. F. Arcelli, I. Pigazzini, R. Roveda, and M. Zanoni. "Automatic Detection of Instability Architectural Smells", International Conference on Software Maintenance and Evolution, (ICSME'16), USA, pp. 433–437, 2-7 October 2016.

[7] V. R., Basili, G. Caldiera, and H. D. Rombach, "Goal Question Metric Paradigm", Encyclopedia of Software Engineering, John Wiley & Sons, pp. 528-532, 1994.

[8] O. Baysal, R. Holmes, and M. W. Godfrey, "Developer Dashboards: The Need for Qualitative Analytics", IEEE Software, IEEE, 30 (4), pp. 56-52, July-Aug. 2014.

[9] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, E. Lim, A. MacCormack, R. Nord, I. Ozkaya, R. Sangwan, C. Seaman, K. Sullivan, and N. Zazworka, "Managing Technical Debt in Software-Reliant Systems", Workshop on Future of software engineering research (FoSER '10), pp. 47-52, New Mexico, USA, 07 - 08 November 2010.

[10] A. Chatzigeorgiou, Ap. Ampatzoglou, Ar. Ampatzoglou, and T. Amanatidis, "Estimating the breaking point for technical debt", 7th International Workshop on Managing Technical Debt (MTD' 15), IEEE, Germany, pp.53-56, Oct. 2015.

[11] S. Charalampidou, A. Ampatzoglou, A. Chatzigeorgiou, A. Gkortzis, and P. Avgeriou, "Identifying Extract Method Refactoring Opportunities Based on Functional Relevance" IEEE Trans. Software Eng., 43(10), pp. 954-974, 2017.

[12] J.-C. Chen, and S.-J. Huang, "An empirical analysis of the impact of software development problem factors on software maintainability," Journal of Systems and Software, 82(6), pp. 981–992, 2009.

[13] G. Digkas, M. Lungu, P. Avgeriou, A. Chatzigeorgiou and A. Ampatzoglou, "How do developers fix issues and pay back technical debt in the Apache ecosystem?", 25th International Conference on Software Analysis, Evolution and Reengineering (SANER'18), IEEE, Campobasso, Italy, 20-23 March 2018.

[14] R. J. Eisenberg "A threshold based approach to technical debt", ACM SIGSOFT Software Engineering Notes, 37 (2), pp. 1 - 6, ACM, 2012.

[15] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure It? Manage It? Ignore It? Software Practitioners and Technical Debt", 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE'15), pp. 50-60, Italy, 30 August - 04 September 2015.

[16] A. Field, "Discovering Statistics using IBM SPSS Statistics", SAGE, 2013.

[17] D. D. Galorath, "Software total ownership costs: development is only job one," Software Tech News, 11(3), 2008.

[18] L. Heinemann, B. Hummel, and D. Steidl, "Teamscale: Software Quality Control in Real-Time", 36th International Conference on Software Engineering (ICSE'14), Companion, pp. 592-595, Hyderabad, India, 31 May – 07 June 2014.

[19] ISO/IEC 9126-1:2001, Software engineering - Product quality (Part 1: Quality model), Geneva, Switzerland, 2001.

[20] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research part 2: designing a survey", ACM SIGSOFT Softw. Engineering Notes 27(1), pp. 18-20, 2002.

[21] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research: part 3: constructing a survey instrument", ACM SIGSOFT Softw. Engineering Notes 27(2), pp. 20-24, 2002.

[22] B. A. Kitchenham and S. L. Pfleeger, "Principles of survey research part 6: data analysis", ACM SIGSOFT Software Engineering Notes 28(2), pp. 24-27, 2003.

[23] P. Kruchten, R. L. Nord, and I. Ozkaya, "Technical Debt: From Metaphor to Theory and Practice", IEEE Software, 29 (6), 18-21, 2006.

[24] Z. Li, P. Avgeriou, and P. Liang, "A systematic mapping study on technical debt and its management", Journal of Systems and Software, Elsevier, v. 101, pp. 193-220, March 2015.

[25] A. Martini and J. Bosch, "Towards Prioritizing Architecture Technical Debt: Information Needs of Architects and Product Owners", 41st Euromicro Conference on Software Engineering and Advanced Applications, IEEE, Funchal, Portugal, 26-28 Aug. 2015

[26] M. Fokaefs, N. Tsantalis, E. Stroulia, and A. Chatzigeorgiou, "Identification and application of Extract Class refactorings in object-oriented systems", Journal of Systems and Software, Elsevier, 85 (10), pp. 2241-2260, 2012.

[27] J. L. Letouzey and T. Coq, "The SQALE Analysis Model: An Analysis Model Compliant with the Representation Condition for Assessing the Quality of Software Source Code", 2nd International Conference on Advances in System Testing and Validation Lifecycle (VALID' 10), IEEE, pp. 43-48, France, 22-27 Aug. 2010.

[28] E. Lim, N. Taksande, and C. Seaman, "A Balancing Act: What Software Practitioners Have to Say about Technical Debt", IEEE Software, IEEE Computer Society, 29 (6), pp. 22-27, Nov.-Dec. 2012.

[29] A. Martini, T. Besker, J. Bosch, "Technical Debt tracking: Current state of practice: A survey and multiple case study in 15 large organizations", *Science of Computer Programming*, Elsevier, 163 (1), pp. 42-61, October 2018.

[30] S. L. Pfleeger, B. A. Kitchenham, "Principles of survey research: part 1: turning lemons into lemonade", ACM SIGSOFT Software Engineering Notes, 26(6), pp. 16-18, 2001.

[31] S. M. A. Shah, J. Dietrich, C. McCartin, "Making Smart Moves to Untangle Programs", 16th European Conference on Software Maintenance and Reengineering (CSMR 2012), Szeged, Hungary, IEEE Computer Society, 72–30 March 2012.

[32] C. Seaman and Y. Guo, "Measuring and monitoring technical debt", Advances in Computers, Elsevier, v.82, pp. 25 - 46, 2011.

[33] D. Steidl, F. Deissenboeck, M. Poehlmann, R.Heinke, and B. Uhink-Mergenthaler, "Continuous Software Quality Control in Practice", International Conference on Software Maintenance and Evolution (ICSME'14), Canada, 2014.

[34] H. van Vliet, "Software Engineering: Principles and Practice", John Wiley, 2008.