e-Informatica Software Engineering Journal, Volume 14, Issue 1, 2020, pages: 61–76, DOI 10.37190/e-Inf200102

Technical Debt Aware Estimations in Software Engineering: A Systematic Mapping Study

Paweł Klimczyk*, Lech Madeyski**
*GEMOTIAL
**Faculty of Computer Science and Management, Wroclaw University of Science and Technology, Wyb.Wyspianskiego 27, 50-370 Wroclaw, Poland
pawel@klimczyk.pl, lech.madeyski@pwr.edu.pl

Abstract

Context: The Technical Debt metaphor has grown in popularity. More software is being created and has to be maintained. Agile methodologies, in particular Scrum, are widely used by development teams around the world. Estimation is an often practised step in sprint planning. The subject matter of this paper is the impact technical debt has on estimations.
Objective: The goal of this research is to identify estimation problems and their solutions due to previously introduced technical debt in software projects.
Method: The Systematic mapping study (SMS) method was applied in the research. Papers were selected from the popular digital databases (IEEE, ACM, Scopus, etc.) using defined search criteria. Afterwards, a snowballing procedure was performed and the final publication set was filtered using inclusion/exclusion criteria.
Results: 42 studies were selected and evaluated. Five categories of problems and seven proposed solutions to the problems have been extracted from the papers. Problems include items related to business perspective (delivery pressure or lack of technical debt understanding by business decision-makers) and technical perspective (difficulties in forecasting architectural technical debt impact or limits of source code analysis). Solutions were categorized in: more sophisticated decision-making tools for business managers, better tools for estimation support and technical debt management tools on an architectural-level, portfolio approach to technical debt, code audit and technical debt reduction routine conducted every sprint.
Conclusion: The results of this mapping study can help taking the appropriate approach in technical debt mitigation in organizations. However, the outcome of the conducted research shows that the problem of measuring technical debt impact on estimations has not yet been solved. We propose several directions for further investigation. In particular, we would focus on more sophisticated decision-making tools.

Keywords: Software estimation, technical debt, project management, decision making, change impact

1. Introduction

Today software is present in all industries worldwide. The Industry 4.0 [1, 2][1] or Internet of Things [3] concepts are based on software to operate and provide solutions. Agile methods were proposed to *better handle inevitable changes* [4]. A number of practices have become popular, e.g., Continuous Integration, TDD, Pair Programming, to ensure sufficient production code and tests quality (e.g., [5–7]) and software development productivity (e.g., [8]).

Cunningham [10] introduced the *technical debt* term to describe shortcuts taken by soft-

[1]Note that two reference lists are included at the end of this paper: the first one includes papers found during our systematic mapping, the second one is the main reference list.

Submitted: 10 March 2019; Revised: 20 January 2020; Accepted: 22 January 2020; Available online: 13 February 2020

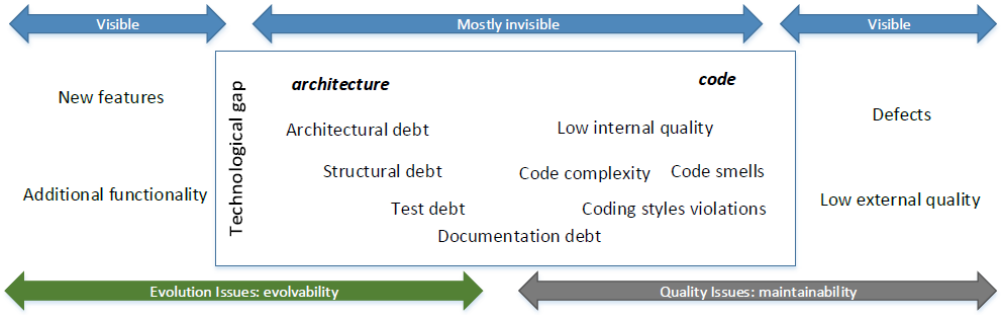

Figure 1. Technical Debt Landscape (inspired by [9])

ware engineers in order to deliver value on time. "A little debt speeds development so long as it is paid back promptly with a rewrite.... The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt." [10]. The number of software developers increases every year. That implies creating more code and more technical debt in the result. According to Google Trends[2] technical debt metaphor has been growing in popularity.

Software project features may be delivered faster to users, but the effects of taking technical debt (e.g., storing application data in a file instead of a database) will have to be addressed in the future. As stated by Fowler [11], technical debt can be taken intentionally or unintentionally. Along with technical debt there is a *interests* concept. *Interests* can be considered as "the extra maintenance cost spent for not achieving the ideal quality level." [12]. It is a metaphor for unpaid technical debt becoming more expensive to repay over time. Technical debt grows during the software development process as stated in [13].

Technical Debt Landscape (Figure 1) was introduced by Kruchten et al. [9]. The landscape identifies *mostly invisible* area where potential problems affecting estimations exist. *Mostly invisible* items are hidden to everybody apart from software engineers. Other members of the project team are aware of them, but might not know the details. The authors state: *Technical debt should not be treated in isolation from adding new functionality or fixing defects* and *The challenge is in expressing all software development activities in terms of sequences of changes associated with a cost and a value* [9]. Software development teams should communicate the "technological gap" in effort estimation so *mostly invisible* parts are known to the managers and stakeholders.

Estimation is a process of rough calculation of how much time is needed to deliver business value related to the estimated task or feature. There is a number of techniques helping developers to provide more accurate estimation [14–16] (e.g., poker planning, smart use cases or bucket system). Some of them use the developer's experience in a project to consider technical debt impact on estimation accuracy.

Estimations are straightforward in well-specified projects. Development teams start from scratch and will introduce technical debt. As new features are implemented or as existing features are extended, the project's complexity increases. The problem with estimations becomes visible after the technical debt has been taken and has to be addressed. It may be expected that forecasting technical debt impact on a new or changed feature is more difficult in later development stages. Estimations are becoming inaccurate and one of the reasons is improper technical debt measurement. The problem has to be addressed.

The goal of this research was to conduct a systematic mapping study on technical debt in the context of estimations. A number of publications were collected, examined and categorized giving several directions for further research.

The paper is organized as follows: Section 2 presents related work. Section 3 defines research

[2]https://trends.google.com/trends/explore?date=all&q=technicaldebt

questions for this systematic mapping study (SMS). Research methodology and crucial details of the SMS protocol are described in Section 4. Section 5 shows study results with a detailed description. In Section 6 we interpret responses to the posed study research questions. Section 7 presents threats to validity, while in Section 8 we conclude the work and show directions of further research. A list of primary sources found in our SMS is presented before references.

2. Related work

The amount of produced software worldwide increases every year which in turn affects technical debt. A number of studies have been conducted to address the problem of increasing technical debt from various perspectives.

Fernández-Sánchez et al. [17] searched for elements required in the technical debt management. They came up with a list of 12 items that will support decision making in managing technical debt. Items are divided into three types:*(T1) Basic decision-making factors, (T2) Cost estimation techniques and (T3) Practices and techniques for decision-making.* The result of this article is a framework introduced to aid decision making in technical debt management.

Another research by Fernández-Sánchez et al. [18] covers available techniques and methods for technical debt management from a software architecture perspective. In their systematic mapping study authors discovered the impact of various technical debt types, like code technical debt, documentation technical debt etc. on architectural technical debt. The conclusion is that further studies on architectural debt from a more holistic approach are needed.

Ribeiro et al. [19] provides a list of 14 decision criteria on which technical debt repayment can be prioritized. Authors conclude that none of the researched studies has performed an empirical evaluation. In the authors' opinion, this may indicate a low level of maturity in decision-making criteria itself.

Li et al. [20] in their mapping study on technical debt and its management identify a list of ten technical debt types and 29 tools used as technical debt management systems. They indicate, however, that only four tools are dedicated to technical debt management. The rest is adapted in various ways from other software development areas. They conclude that there is a need for more sophisticated and dedicated technical debt management tools and further research on technical debt management. More high-level studies should be conducted by the software engineering community.

In another related work, Behutiye et al. [21] analyse the concept of technical debt in Agile Software Development (ASD). A list of ten causes and five consequences of incurring technical debt in ASD was identified in the research. Authors also classified a list of technical debt management strategies in ASD. The research indicates *the need for more tools, models and guidelines that support management of technical debt in ASD* [21] and the role of architecture in ASD.

The financial aspect is considered by Ampatzoglou et al. [22]. Authors introduced a glossary of financial terminology and classification schema of financial approaches used in technical debt management. The publication also states that it is easier for developers to communicate with non-technical managers.

Systematic mapping study on identification and management of technical debt was conducted in [23]. Research enumerates strategies that have been proposed to identify or manage technical debt in software projects. The conclusion is that most of the strategies are new but they lack studies to evaluate their real applicability.

None of the mentioned publications addressed the problem of technical debt impact on estimations. The goal of our work differs from the other secondary studies in terms of the research perspective and scope. Our study focuses on understanding how task delivery estimation is affected by technical debt and what software development teams do to develop software according to plan.

3. Research objectives

The objectives of this study were to identify problems in estimations due to existing technical debt

in software projects and collect ideas on how development teams try to overcome the problems. Following research questions were stated:

RQ1: What are the problems for the development team during task estimation due to technical debt?

The purpose of this question is to confirm problem existence. Potentially it could be possible to identify groups of similar problems.

RQ2: What kind of solutions are proposed to mitigate the impact of technical debt on task estimation?

The purpose of this question is to collect the actions taken by development teams to reduce technical debt factor in estimations.

4. Research methodology

In software engineering, guidelines developed by Kitchenham et al. [24] and Petersen et al. [25] provide comprehensive instructions on how to conduct systematic literature reviews (SLR) and systematic mapping studies. They share some commonalities (e.g., related to searching and study selection). However, the difference between both approaches is that systematic literature reviews focus on synthesising the evidence and gaining a new knowledge, while systematic mapping studies [25] are focused on structuring the research area and creating an overview. Systematic mapping study was chosen as a framework for this research to answer the questions posed in Section 3.

4.1. Systematic Mapping Study (SMS) protocol

Our protocol defines the procedures we intended to use for SMS including the following steps:

1. Define study objectives and research questions
2. Define search query and digital source databases
3. Define publication selection criteria
4. Define inclusion and exclusion criteria
5. Conduct data extraction and assessment
6. Conduct data synthesis

After trialling the specified processes, the final version of the protocol was agreed by both authors. The following sections are based on the processes defined in the protocol. However, it is worth mentioning that we have added an additional exclusion criteria (short summary reports) that was not mentioned in the protocol.

4.2. Search query

We performed a series of trial queries against electronic databases. In result the following search query was formulated:

("software") AND ("technical debt" OR "change impact") AND ("estimation" OR "decision making" OR "management")

Such a search query will find publications with a technical debt aspect in various contexts.

4.3. Digital source databases

Publication sources include all popular academic databases. The year 1992 was chosen as the timeframe limit since Cunningham published his paper at that time [10]. Studies from following digital source databases were included:

- IEEE Xplore [26]
- ACM Digital Library [27]
- Springer Link [28]
- Science Direct [29]
- Scopus [30]

4.4. Inclusion/exclusion criteria

Search query defined in Section 4.2 returned a total number of 2003 candidate documents for primary studies set. The distribution of documents per source database is presented in Table 1.

Primary studies set contained many irrelevant publications, due to query search generic nature. Thus, following inclusion/exclusion criteria were applied to select only relevant studies.

Inclusion criteria:

- Publications that describe the problem of technical debt in software development and technical debt management.
- Case studies and surveys based on industrial examples of technical debt management.

Table 1. Distribution of publications per source

Source	No. of publications returned by search query	No. of publications included in our paper
IEEE Xplore	275	14
ACM Digital Library	652	10
Springer Link	341	5
Science Direct	369	5
Scopus	366	7
Snowballing	n/a	1

- Technical debt management technique proposals.
- Papers written since 1992 when Cunningham [10] introduced the *technical debt* term[3].
- Papers written in English – English is a common language used by researchers.

Exclusion criteria:

- Publications that only mention technical debt as an issue, but do not focus on deeper elaboration/description of the problem.
- Short summary reports about what workshop participants discussed instead of real research contributions – short summaries do not provide enough information.
- Duplicate publications.
- Publications with only abstract available – we were interested in the details of a particular research.
- Papers not written in English.

4.4.1. Snowballing

The importance of the snowballing step in SMS is described in [31]. Backward snowballing was performed for this study. Papers found in snowballing were checked using the same inclusion/exclusion criteria list as primary papers. The snowballing technique found one additional publication.

4.5. Data extraction and assessment

Data extraction and assessment process focused on collecting evidence that can formulate an answer to RQs. All filtered publications were read in full. Microsoft Excel was used to record and organize the following data: title, source, citation eligible for RQ1 or RQ2 and publication type. The assessment was based on whether a study provides evidence to answer one of the RQs.

4.6. Initial research set

Initial research set consisted of 45 articles. After applying inclusion/exclusion criteria papers [S1], [S2] and [S3] was excluded. Decisions were discussed by both authors.

4.7. Rigor and relevance

We applied a checklist proposed by Ivarsson and Gorschek [32] to access rigor and relevance of the final dataset. The rating model consists of two perspectives to measure: rigor and relevance. Rigor refers to how an evaluation is performed and how is it reported. Relevance measures the industrial applicability in the usage context, used research method, subjects/users and scalability. Each item is scored by 0, 0.5 and 1 in rigor perspective and 0 or 1 in relevance perspective.

The first author rated the studies for quality assurance. The rigor and relevance scores distribution in our SMS is presented in Figure 2.

In order to review the selection agreement among the authors, a Kappa analysis [33] was performed. Seven randomly selected[4] publications were examined by the second author. Based on the selected sample Kappa value was calcu-

[3]The technical debt knowledge, along with programming languages, has evolved over last 30 years, and we do not expect that problems and solutions discussed in papers written before 1992, and not cited after that year, would add value to the paper.

[4]https://www.random.org/sequences/?min=1&max=42&col=1&format=html&rnd=new

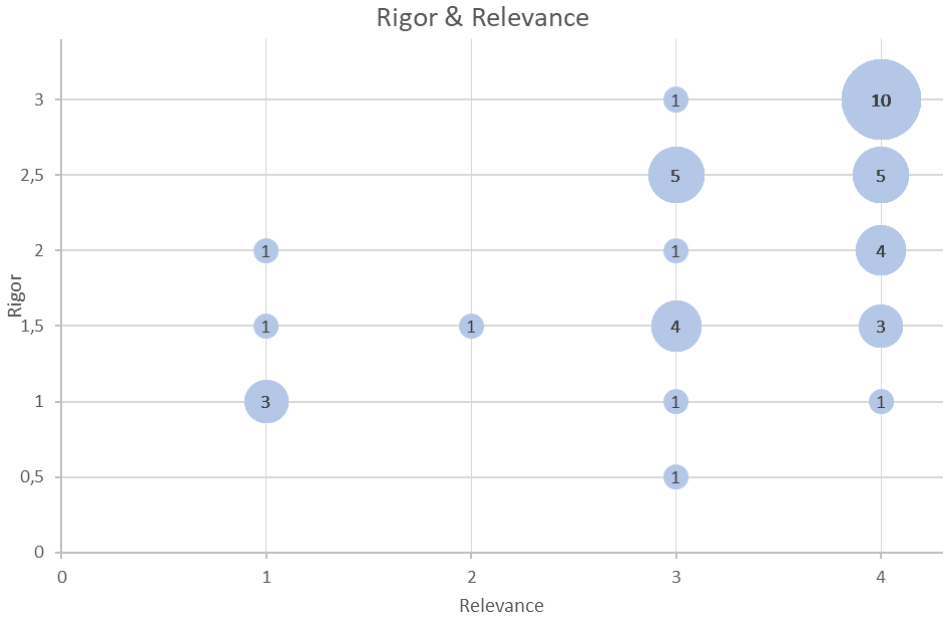

Figure 2. Mapping of selected papers with respect to rigor and relevance

lated – the strength of agreement was very good ($\kappa = 1.0$).

4.8. Final set of papers

We selected 42 publications, see Table 2 and the list of primary studies found in our systematic mapping, presented before references. 41 of the papers were filtered through digital source databases using search query presented in Section 4.2. An additional one was found during the snowballing process. Table 1 presents a distribution of publications per digital source databases and snowballing procedure. It is worth mentioning that case studies were the most popular publication types among the accepted primary studies.

At this point we assessed all evidence for eligibility and divided into two groups: **Identified problem categories (G1 – addressing RQ1)** and **Identified potential solution categories (G2 – addressing RQ2)**. Groups would later provide potential answers to RQs accordingly. The next step was to synthesise the data.

4.9. Data synthesis

The purpose of data processing is to synthesize extracted data in order to answer RQs from Section 3. Data extracted in Section 4.5 was divided into two groups. Each group contains a number of categories that emerged from examined publications. Category names were deduced from clustering items in each group.

Each category has its description and several papers addressing a particular subject. Results of data synthesis are available in Table 2.

5. Study results

We conducted a systematic mapping study according to the procedure described in Section 4. In total 42 publications were examined. During data extraction and synthesis, five categories of problems (corresponding to RQ1) and seven categories of proposed solutions (corresponding to RQ2) to the problems were identified for the selected studies. RQs findings are discussed in Sections 5.1 and 5.2.

Table 2. Data synthesis results

	Category	Description	No. of studies	Sources
Identified problem categories for development teams during estimations (G1)	Business pressure on delivery	Studies showing business pressure of any kind on the project delivery (e.g., release project ahead of competition, new regulations introduced by public administration, raising company market value)	11	[S4], [S5], [S6], [S7], [S8], [S9], [S10], [S11], [S12], [S13], [S14]
	Lack of technical debt awareness in company	Studies showing that non-technical stakeholders are now aware of technical debt impact on estimations	5	[S11], [S15], [S16], [S17], [S18]
	No procedures for technical debt management	Studies stating a lack of any technical debt management techniques incorporated in software engineering process	3	[S16], [S19], [S20]
	Architectural technical debt impact	Studies providing samples where architecture technical debt had impact on software estimation and delivery	9	[S8], [S9], [S15], [S16], [S21], [S22], [S23], [S24], [S25]
	Source code analysis is not sufficient	Studies claiming that sole code analysis measurements are not enough in task estimation improvements	7	[S9], [S15], [S16], [S21], [S26], [S27], [S28]
			Total distinct studies	25
Identified proposed solution categories to mitigate technical debt in estimations problem (G2)	Tools for decision support	Studies indicating need of high level tools that will help business people to take development decision with technical debt consideration (e.g., which parts of the system will be affected by implementing particular feature, how much human resources needs to be involved)	14	[S7], [S8], [S9], [S11], [S29], [S30], [S31], [S32], [S23], [S33], [S34], [S24], [S18], [S35]
	Tools for estimation support	Studies stating the need of technical debt estimation tool for development team. Such tool would improve estimation accuracy	9	[S17], [S15], [S27], [S36], [S37], [S38], [S39], [S40], [S20]
	Portfolio approach (technical debt Items)	Studies proposing various catalogues of technical debt items managed by development team in structured manner. Newly introduced technical debt should be added to catalog	11	[S5], [S41], [S22], [S28], [S29], [S32], [S42], [S43], [S18], [S35], [S44]
	Architecture level technical debt visualization tool	Studies stating the need of managing technical debt on architectural level. Overview tool of a complex system that would show a map of potentially affected areas by new changes	7	[S21], [S15], [S27], [S9], [S16], [S22], [S24]
	Technical debt reduction in every sprint	Studies suggesting that a certain amount of time should be devoted to reducing technical debt by the development team	8	[S6], [S8], [S5], [S19], [S32], [S45], [S13], [S44]
	Code audit activity	Studies advising to conduct structured code audit periodically	3	[S6], [S19], [S12]
	Extra resources	Studies claiming that more resources such as people, infrastructure or budget are needed	2	[S45], [S22]
			Total distinct studies	37

5.1. Problems in estimation due to technical debt (RQ1)

We gathered five categories of problems in user-story estimation due to technical debt:

- **Business pressure on delivery** – 11 papers (i.e., 44% of publications that identified problems) emphasised that business pressure was the key factor in estimations and therefore technical debt introduction. Hence, we think that this problem is widespread. In one of the studies, authors say: *The participants commonly acknowledged that technical debt is essentially a balance between software quality and business reality* [S6]. Authors list a number of reasons behind that statement: (1) *being contractually obligated to deliver the system under a tight deadline*, (2) *meeting deadlines to integrate with a partner product before release*, (3) *delivering in time for an upcoming trade show that presented food marketing opportunities*, (4) *developing a working prototype to secure investors funding* [S6].
- **Lack of technical debt awareness in company** – Five studies notice that non-technical stakeholders were unaware of technical debt impact on the project. In one study authors write: *From developer's perspective, management remains largely unaware of technical debt and the value of managing it* [S15].
- **No procedures for technical debt management** – Authors of three publications inform about lack of any methodology in projects they have investigated. In one study we can find a statement: *Neither of the product lines had any specific approach for dealing with technical debt management and reduction* [S19].
- **Architectural technical debt impact** – Nine studies conclude that complex code architecture structure and its technical debt has an impact on estimations. Authors of one of the studies stated: *Architectural issues are the greatest source of technical debt... Architectural issues are difficult to deal with, since they were often caused many years previously* [S15].
- **Source code analysis is not sufficient** – This problem is brought by seven studies. Software engineers see that source code analysis does not show the whole picture of the system. This has an impact on estimations. One of the studies stated: *... technical debt is not only about code and code quality. Code analysis tools will identify a small number of black elements. Therefore, code analysis tools aren't sufficient for identifying technical debt...* [S26][5]. In another study, authors write: *Tools do not capture the key areas of accumulating problems in technical debt* [S15].

5.2. Proposed solutions to mitigate the impact of technical debt on task estimation (RQ2)

Conducted research provides seven techniques for mitigating the impact of technical debt on estimations:

- **Tools for decision support** – This finding uncovers a communication gap between organisation units in an organisation. What was not expected was how widespread is the opinion that non-technical management should have a tool for better decision support in the project. As much as 14 of 37 papers (i.e., 38% of papers that identified solutions) emphasised such need.
- **Tools for estimation support** – Nine papers propose introducing estimation support tools for development teams. Authors of one study say *... by the later stages of the project the algorithm is more reliable than manual Planning Poker estimates and thus suitable as a tool for augmenting human effort estimation* [S36].
- **Portfolio approach (technical debt Items)** – As much as 11 of 37 papers that identified solutions propose managing technical debt in a structured way. Developers should fill "technical debt Item" cards so the team is aware of how much technical debt there is in the system. In one of the papers, authors write: *... managers*

[5]The black element refers to technical debt which was visually presented in Figure 2 on page 20 [S26]

expressed that the backlog would be used in the future... to reduce technical debt in small iterations [S22].

- **Architectural-level technical debt visualisation tool** – Seven publications indicate the need for a high-level technical debt monitoring tool. A tool that will have the knowledge about technical debt not only in separate system components but also between them and the system as a whole. Authors of one study stated: *Making the architectural debt visible provides the necessary information for making informed decisions for managing the potential impact of rework over time* [S21]. This issue is also mentioned by others: *The lack of tool support for accurately managing and tracking architectural sources of debt is a key issue...* [S15].
- **Technical debt reduction in every sprint** – Eight publications propose continuous technical debt reduction during every sprint. A related excerpt in one of the papers is as follows: *one participant described a policy of allocating 5 to 10 per cent of each release cycle to addressing technical debt* [S6].
- **Code audit activity** – Three papers ([S6], [S19], [S12]) propose periodical and systematic code audit actions conducted by the development team. Authors of one of the studies conclude: *... conduct audits with the entire development team to make technical debt visible and explicit; track it using a Wiki, backlog, or task board* [S6]
- **Extra resources** – Two papers propose adding extra resources such as people [S22], infrastructure or budget [S45] to the project. Such solutions may indicate a tight project schedule or an attempt to reaching the project deadline.

6. Discussion

The overall goal of this research was to identify problems, as well as proposed solutions occurring in estimations due to previously introduced technical debt. In this section we will present our interpretation of systematic mapping study results and their implications for academia and industry.

6.1. Problems in estimation due to technical debt

Business pressure on delivery and **lack of technical debt awareness in management** are related to the business perspective in a particular software project. The main purpose of building software is to support other processes. Managers and business officers are focused on growing the organization. Software support can give them a competitive advantage and that is why they force pressure on short software release cycles.

No procedures for technical debt management mentioned in three research papers indicate immature development process. This may be due to various reasons. Company owners may not be aware of the technical debt problem or may consider a particular project as a prototype where technical debt is not considered as a problem. On the other side, the project can be so big that introducing new development procedures is too cumbersome or too expensive. Finally, the development team may not know how to introduce such procedures.

Results such as **architectural technical debt impact** and **source code analysis is not sufficient**, can be interpreted differently. Those problems are more related to technical aspects. The **architectural technical debt impact** item is strongly bound to project evolution. For instance, the mainstream in web development is moving to cloud-based solutions and application containers providing better scalability and flexibility. Adjusting old software can be difficult and can be considered as a sample of architectural technical debt. **Source code analysis is** also **not sufficient** because engineers would adjust their code in such a way that it will pass the code analysis, but remind a poor quality.

Depending from which perspective we consider the situation different problems are present. In the worst-case scenario, all of them can occur in the organization and will slow down the development process even further.

6.2. Solutions to mitigate technical debt in estimations

Only one proposed solution focuses on non-technical stakeholders (**tools for decision support**). However, 38% of examined studies (14 of 37) state that this is the desired solution. This indicates the complicated nature of modern software solutions. Managers and decision-makers have difficulties understanding the technical implications of their business decisions. Especially in competitive markets, where the software should be adjusted quickly, managers should see the results fast and be able to respond to them. Worth mentioning here are automatic code generators where solutions can be created without software engineers.

Another interesting interpretation arises from **portfolio approach (technical debt items)**, **technical debt reduction in every sprint** and **code audit activity**. All of those solutions can be concluded as a need for deeper software development processes standardization and/or regulations. In other industries like medicine, maritime, aviation or automotive rules and regulations according to which certain procedures have to be conducted do exist. In IT there is ISO 25010 standard, but it is not mandatory to implement it.

The findings indicate that "Time To Market" has the biggest impact on schedule and the decision to repay or not the technical debt. The software solutions are too complicated and cannot be adapted fast enough in a rapidly changing world. An interesting fact the study uncovers is that source code analysis tools are not sufficient to cope with technical debt in estimations.

Based on the information from the performed SMS, we recommend focus future research on various decision-support levels. The complexity of software solutions grows and it is more difficult to get an overview from both business and technical perspectives. We propose that such decision-support research should take in consideration software maintenance and evolution.

7. Threats to validity

A systematic mapping study is conducted by people and thus an inevitable risk is related to the bias that may come from the choice of search engines/digital libraries and of search terms that may favour finding some studies and perhaps missing others. Hence, an important threat to the validity of this SMS is related to the search strategy employed and the possibility that we have not identified all relevant papers. The completeness of the search depends on the search string used, the scope of the search in terms of selected search engines, as well as their limitations Brereton et al. [34]. For example, it is possible to extend the search query even further by adding additional words like "managing". We do not think this is a significant threat. Nevertheless, it is still possible that after such extension the result set of papers would be a different, but (in our opinion) to a minor extent. To reduce this threat we selected a range of digital libraries and thus widened its scope. We also used a known set of references to validate the search terms before undertaking the mapping study and the search terms were amended where necessary (e.g., we included "change impact" that we initially missed).

The time window chosen by us (since 1992 till now) can be seen as a threat. That said, we think that the knowledge about technical debt, software development and programming languages has evolved to such extent that we probably do not lose anything crucial excluding papers before 1992.

We also conducted snowballing to limit the possibility of missing relevant papers. Only one additional paper was identified by searching the references of included studies.

A closely related threat is that "grey literature" may not be found due to the nature of digital libraries used. Snowballing can be seen as a partial solution to limit this threat as references of the papers found in digital libraries may include "grey literature" as well.

It is also worth mentioning that categories synthesised from publications data extraction emerged from our best understanding of the topic. We proposed category names presented in Table 2 based on our experience in software engineering.

We limited the scope of our search to articles written in English. Thus the presented results can be biased by omitting publications written

Table 3. Evaluation of our mapping process (see [25])

Rubric	Score	Description
Need for review	1	Partial evaluation – motivations and questions are provided.
Choosing the search strategy	1	Minimal evaluation – two search strategies (automated database search and snowballing) have been used.
Evaluation of the search	2	Partial evaluation – at least one action has been taken to improve the reliability of the search and the inclusion/exclusion.
Extraction and classification	2	Partial evaluation – at least one action has been taken to increase the reliability of the extraction process, and research type and method have been classified.
Study validity	1	Full evaluation – threats and limitations are described.

in other languages (e.g., Chinese). However, we based our research on the most popular language among software engineering researchers and practitioners.

A search-related limitation of this mapping study is that the search only covers publications that were included in the chosen digital libraries before January 2019. This date is related to the moment when the mapping study was performed. It is therefore probable (due to the fact that technical debt is perceived as an interesting topic) that a number of other relevant papers will have been published since this date that we have not included in this mapping study. However, this limitation is difficult to avoid and the common solution is to conduct a new search and/or snowballing to update the results of the mapping study.

Additionally, Table 3 presents an evaluation of our mapping process on a basis of the quality checklist rubric criteria (defined by Petersen et al. [25]) including: identifying the need for SMS, study identification, data extraction and classification, as well as validity discussion.

8. Conclusions

In this systematic mapping study, 42 out of 2003 relevant publications were selected. 41 from query search in five digital databases and one additional from the snowballing procedure. The contribution of this study is a categorisation of technical debt related issues in task estimations and proposed solutions to the issues presented in Section 5. Five problems and seven solutions identified in literature have been categorised. Furthermore, the majority of identified categories of problems and solutions include real-life examples describing industry cases.

The technical debt impact on task estimation is an important issue to address. Our SMS shows seven approaches to extend the current state of technical debt management. We conclude that the task estimation accuracy can be further improved in one of the following directions:

- business direction – research on how the managers can gain more insight into the software system that is supporting their business. Understand the system's current limitations and the impact of new business decisions on it. That implies research on how software engineers can improve communication with "the business."
- operational direction – research on software systems maintainability and development routines. That includes new ways of formalizing and structuring software components, data flows, integrations and others so that it would be easy to analyse new requirements impact on the software project.

The problem of business pressure on features delivery has appeared in our findings on several occasions. Our further research will focus on decision-making tools. In our opinion, there is a room for improvement that will potentially help development teams to measure the impact of technical debt on estimations with more accurate precision.

References found during our systematic mapping study

[S1] 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," in *Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, FoSER '10. ACM, 2010, pp. 47–52.

[S2] C. Izurieta, I. Ozkaya, C. Seaman, P. Kruchten, R.L. Nord, W. Snipes, and P. Avgeriou, "Perspectives on managing technical debt: A transition point and roadmap from dagstuhl," in *Joint Proceedings of the 4th International Workshop on Quantitative Approaches to Software Quality (QuASoQ 2016) and 1st International Workshop on Technical Debt Analytics (TDA 2016)*, 2016, pp. 84–87. [Online]. http://ceur-ws.org/Vol-1771/paper15.pdf

[S3] P. Kruchten, R.L. Nord, I. Ozkaya, and D. Falessi, "Technical debt: Towards a crisper definition report on the 4th international workshop on managing technical debt," *ACM SIGSOFT Software Engineering Notes*, Vol. 38, No. 5, 2013, pp. 51–54.

[S4] Y. Guo, C. Seaman, R. Gomes, A. Cavalcanti, G. Tonin, F.Q.B. Da Silva, A.L.M. Santos, and C. Siebra, "Tracking technical debt – An exploratory case study," in *Proceedings of the 27th IEEE International Conference on Software Maintenance*, ICSM '11. IEEE Computer Society, 2011, pp. 528–531.

[S5] K. Power, "Understanding the impact of technical debt on the capacity and velocity of teams and organizations: Viewing team and organization capacity as a portfolio of real options," in *Proceedings of the 4th International Workshop on Managing Technical Debt*, 2013, pp. 28–31.

[S6] E. Lim, N. Taksande, and C. Seaman, "A balancing act: What software practitioners have to say about technical debt," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 22–27.

[S7] T. Klinger, P. Tarr, P. Wagstrom, and C. Williams, "An enterprise perspective on technical debt," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 35–38.

[S8] Z. Codabux and B. Williams, "Managing technical debt: An industrial case study," in *Proceedings of the 4th International Workshop on Managing Technical Debt*, MTD '13. IEEE Press, 2013, pp. 8–15.

[S9] A. Martini, J. Bosch, and M. Chaudron, "Investigating architectural technical debt accumulation and refactoring over time," *Information and Software Technology*, Vol. 67, No. C, 2015, pp. 237–253.

[S10] N. Ramasubbu, C.F. Kemerer, and C.J. Woodard, "Managing technical debt: Insights from recent empirical evidence," *IEEE Software*, Vol. 32, No. 2, 2015, pp. 22–25.

[S11] J. Bohnet and J. Döllner, "Monitoring code quality and development activity by software maps," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 9–16.

[S12] J.C. Rocha, V. Zapalowski, and I. Nunes, "Understanding technical debt at the code level from the perspective of software developers," in *Proceedings of the 31st Brazilian Symposium on Software Engineering, SBES*, 2017, pp. 64–73.

[S13] R.K. Gupta, P. Manikreddy, and K.C. Arya, "Pragmatic scrum transformation: Challenges, practices and impacts during the journey A case study in a multi-location legacy software product development team," in *Proceedings of the 10th Innovations in Software Engineering Conference, ISEC*, 2017, pp. 147–156.

[S14] N. Rios, R.O. Spínola, M.G. de Mendonça Neto, and C.B. Seaman, "A study of factors that lead development teams to incur technical debt in software projects," in *Proceedings of the 44th Euromicro Conference on Software Engineering and Advanced Applications, SEAA*, 2018, pp. 429–436.

[S15] N.A. Ernst, S. Bellomo, I. Ozkaya, R.L. Nord, and I. Gorton, "Measure it? Manage it? Ignore it? Software practitioners and technical debt," in *Proceedings of the 10th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2015. ACM, 2015, pp. 50–60.

[S16] J. Yli-Huumo, A. Maglyas, and K. Smolander, "How do software development teams manage technical debt? – An empirical study," *Journal of Systems and Software*, Vol. 120, 2016, pp. 195–218.

[S17] C.Y. Chen, C.W. She, and J.D. Tang, "An object-based, attribute-oriented approach for software change impact analysis," in *Proceedings of the IEEE International Conference on Industrial Engineering and Engineering Management*, 2007, pp. 577–581.

[S18] R.R. de Almeida, U. Kulesza, C. Treude, D.C. Feitosa, and A.H.G. Lima, "Aligning technical debt prioritization with business objectives: A multiple-case study," in *Proceedings of the IEEE International Conference on Soft-*

ware Maintenance and Evolution, ICSME, 2018, pp. 655–664.

[S19] J. Yli-Huumo, A. Maglyas, and K. Smolander, "The sources and approaches to management of technical debt: A case study of two product lines in a middle-size finnish software company," in *Proceedings of the 15th International Conference Product-Focused Software Process Improvement*, 2014, pp. 93–107.

[S20] T. Besker, A. Martini, J. Bosch, and M. Tichy, "An investigation of technical debt in automatic production systems," in *Proceedings of the XP2017 Scientific Workshops*, 2017, pp. 6:1–6:7.

[S21] R.L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," in *Proceedings of the Joint Working IEEE/IFIP Conference on Software Architecture and European Conference on Software Architecture*, WICSA-ECSA '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 91–100.

[S22] J. Yli-Huumo, A. Maglyas, K. Smolander, J. Haller, and H. Törnroos, "Developing processes to increase technical debt visibility and manageability – An action research study in industry," in *Proceedings of the International Conference on Product-Focused Software Process Improvement*, Lecture Notes in Computer Science, Vol. 10027. Springer International Publishing, 2016, pp. 368–378.

[S23] C. de Souza and D. Redmiles, "An empirical study of software developers' management of dependencies and changes," in *Proceedings of the 30th International Conference on Software Engineering*, 2008, pp. 241–250.

[S24] A. Martini, E. Sikander, and N. Madlani, "A semi-automated framework for the identification and estimation of architectural technical debt: A comparative case-study on the modularization of a software component," *Information and Software Technology*, Vol. 93, 2018, pp. 264–279.

[S25] T. Besker, A. Martini, and J. Bosch, "The pricey bill of technical debt: When and by whom will it be paid?" in *Proceedings of the IEEE International Conference on Software Maintenance and Evolution, ICSME*, 2017, pp. 13–23.

[S26] P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.

[S27] Z. Li, P. Liang, and P. Avgeriou, "Architectural technical debt identification based on architecture decisions and change scenarios," in *Proceedings of the 12th Working IEEE/IFIP Conference on Software Architecture*, 2015, pp. 65–74.

[S28] N. Zazworka, R.O. Spínola, A. Vetro, F. Shull, and C. Seaman, "A case study on effectively identifying technical debt," in *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13. ACM, 2013, pp. 42–47.

[S29] C. Seaman, Y. Guo, C. Izurieta, Y. Cai, N. Zazworka, F. Shull, and A. Vetrò, "Using technical debt data in decision making: Potential decision approaches," in *Proceedings of the Third International Workshop on Managing Technical Debt*, MTD '12. IEEE Press, 2012, pp. 45–48.

[S30] C. Fernàndez-Sànchez, J. Garbajosa, and A. Yagüe, "A framework to aid in decision making for technical debt management," in *Proceedings of the 7th IEEE International Workshop on Managing Technical Debt, MTD@ICSME*, 2015, pp. 69–76.

[S31] H. Jason and R. Günther, "When-to-release decisions in consideration of technical debt," in *Proceedings of the Sixth International Workshop on Managing Technical Debt, MTD@ICSME*, 2014, pp. 31–34.

[S32] R.K. Gupta, P. Manikreddy, S. Naik, and K. Arya, "Pragmatic approach for managing technical debt in legacy software project," in *Proceedings of the 9th India Software Engineering Conference*, ISEC '16. ACM, 2016, pp. 170–176.

[S33] A. Pacheco, G. Marín-Raventós, and G. López, "Designing a technical debt visualization tool to improve stakeholder communication in the decision-making process: A case study," in *Proceedings of the 12th IFIP WG 8.9 Working Conference on Research and Practical Issues of Enterprise Information Systems*, 2018, pp. 15–26.

[S34] T. Amanatidis, A. Chatzigeorgiou, and A. Ampatzoglou, "The relation between technical debt and corrective maintenance in PHP web applications," *Information and Software Technology*, Vol. 90, 2017, pp. 70–74.

[S35] M. M. Bomfim and V. A. Santos, "Strategies for reducing technical debt in agile teams," in *Proceedings of the Brazilian Workshop on Agile Methods*. Springer International Publishing, 2017, pp. 60—71.

[S36] K. Moharreri, A.V. Sapre, J. Ramanathan, and R. Ramnath, "Cost-effective supervised learning models for software effort estimation in agile environments," in *Proceedings of the Computer Software and Applications Conference (COMPSAC)*, 2016, pp. 135–140.

[S37] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in

Proceedings of the 2nd Workshop on Managing Technical Debt, MTD '11. ACM, 2011, pp. 1–8.

[S38] B. Tanveer, "Guidelines for utilizing change impact analysis when estimating effort in agile software development," in *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering, EASE*, 2017, pp. 252–257.

[S39] S.J. Kabeer, M. Nayebi, G. Ruhe, C. Carlson, and F. Chew, "Predicting the vector impact of change – an industrial case study at brightsquid," in *Proceedings of the ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM*, 2017, pp. 131–140.

[S40] B. Tanveer, L. Guzmán, and U.M. Engel, "Effort estimation in agile software development: Case study and improvement framework," *Journal of Software: Evolution and Process*, Vol. 29, No. 11, 2017.

[S41] K. Schmid, "A formal approach to technical debt decision making," in *Proceedings of the 9th International ACM Sigsoft Conference on Quality of Software Architectures*, QoSA '13. ACM, 2013, pp. 153–162.

[S42] Y. Guo and C. Seaman, "A portfolio approach to technical debt management," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 31–34.

[S43] Y. Guo, R.O. Spínola, and C. Seaman, "Exploring the costs of technical debt management – A case study," *Empirical Software Engineering*, Vol. 21, No. 1, 2016, pp. 159–182.

[S44] P. Mohagheghi and M.E. Aparicio, "An industry experience report on managing product quality requirements in a large organization," *Information and Software Technology*, Vol. 88, 2017, pp. 96–109.

[S45] Z.S. Hossein Abad, R. Karimpour, J. Ho, S.M. Didar-Al-Alam, G. Ruhe, E. Tse, K. Barabash, and I. Hargreaves, "Understanding the impact of technical debt in coding and testing: An exploratory case study," in *Proceedings of the 3rd International Workshop on Software Engineering Research and Industrial Practice*, SER& IP '16. ACM, 2016, pp. 25–31.

References

[1] E. Mueller, X.L. Chen, and R. Riedel, "Challenges and requirements for the application of Industry 4.0: A special insight with the usage of cyber-physical system," *Chinese Journal of Mechanical Engineering*, Vol. 30, No. 5, 2017, pp. 1050–1057.

[2] M. Hermann, T. Pentek, and B. Otto, "Design principles for Industrie 4.0 scenarios," in *Proceedings of the 49th Hawaii International Conference on System Sciences (HICSS)*, 2016, pp. 3928–3937.

[3] J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, Vol. 29, No. 7, 2013, pp. 1645–1660.

[4] J. Highsmith and A. Cockburn, "Agile software development: The business of innovation," *Computer*, Vol. 34, No. 9, 2001, pp. 120–122.

[5] L. Madeyski, "On the effects of pair programming on thoroughness and fault-finding effectiveness of unit tests," in *Product-Focused Software Process Improvement*, Lecture Notes in Computer Science, J. Münch and P. Abrahamsson, Eds. Springer Berlin Heidelberg, 2007, Vol. 4589, pp. 207–221.

[6] L. Madeyski, "The impact of test-first programming on branch coverage and mutation score indicator of unit tests: An experiment," *Information and Software Technology*, Vol. 52, No. 2, 2010, pp. 169–184.

[7] L. Madeyski, *Test-Driven Development: An Empirical Evaluation of Agile Practice*. (Heidelberg, London, New York): Springer, 2010.

[8] L. Madeyski and Ł. Szała, "The impact of test-driven development on software development productivity – An empirical study," in *Software Process Improvement*, Lecture Notes in Computer Science, P. Abrahamsson, N. Baddoo, T. Margaria, and R. Messnarz, Eds. Springer Berlin Heidelberg, 2007, Vol. 4764, pp. 200–211.

[9] P. Kruchten, R.L. Nord, and I. Ozkaya, "Technical debt: From metaphor to theory and practice," *IEEE Software*, Vol. 29, No. 6, 2012, pp. 18–21.

[10] W. Cunningham, "The WyCash portfolio management system," in *Addendum to the Proceedings on Object-oriented Program-*

ming Systems, Languages, and Applications, OOPSLA '92. New York, NY, USA: ACM, 1992, pp. 29–30.

[11] M. Fowler, "Technical debt quadrant," 2009. [Online]. https://martinfowler.com/bliki/TechnicalDebtQuadrant.html

[12] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," in *Proceedings of the 2nd Workshop on Managing Technical Debt*, MTD '11. ACM, 2011, pp. 1–8.

[13] E. Tom, A. Aurum, and R. Vidgen, "An exploration of technical debt," *Journal of Systems and Software*, Vol. 86, No. 6, 2013, pp. 1498–1516.

[14] M. Cohn, *Agile Estimating and Planning*. Pearson Education, 2005.

[15] S. Hoogendoorn, *This is Agile: Beyond the Basics. Beyond the Hype. Beyond Scrum*. Dymaxicon, 2014.

[16] "The bucket system," 2017. [Online]. http://www.agileadvice.com/wp-content/uploads/2013/07/H10-Estimation-The-Bucket-System.pdf

[17] C. Fernàndez-Sànchez, J. Garbajosa, and A. Yagüe, "A framework to aid in decision making for technical debt management," *2015 IEEE 7th International Workshop on Managing Technical Debt (MTD)*, 2015, pp. 69–76.

[18] C. Fernández-Sánchez, J. Garbajosa, C. Vidal, and A. Yagüe, "An analysis of techniques and methods for technical debt management: A reflection from the architecture perspective," in *Proceedings of the Second International Workshop on Software Architecture and Metrics*, SAM '15. Piscataway, NJ, USA: IEEE Press, 2015, pp. 22–28.

[19] L.F. Ribeiro, M.A.d.F. Farias, M. Mendonça, and R.O. Spínola, "Decision criteria for the payment of technical debt in software projects: A systematic mapping study," in *Proceedings of the 18th International Conference on Enterprise Information Systems*, ICEIS 2016. SCITEPRESS - Science and Technology Publications, Lda, 2016, pp. 572–579.

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

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

[22] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, "The financial aspect of managing technical debt: A systematic literature review," *Information and Software Technology*, Vol. 64, 2015, pp. 52—73.

[23] N.S. Alves, T.S. Mendes, M.G. de Mendonça, R.O. Spínola, F. Shull, and C. Seaman, "Identification and management of technical debt," *Information and Software Technology*, Vol. 70, No. C, 2016, pp. 100–121.

[24] B.A. Kitchenham, D. Budgen, and P. Brereton, *Evidence-Based Software Engineering and Systematic Reviews*. Chapman and Hall/CRC, 2016.

[25] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, Vol. 64, 2015, pp. 1 – 18.

[26] "IEEE Xplore Digital Library," 2017. [Online]. http://ieeexplore.ieee.org

[27] "ACM digital library," 2017. [Online]. http://dl.acm.org

[28] "Springer Link," 2017. [Online]. https://link.springer.com

[29] "Science Direct," 2017. [Online]. http://www.sciencedirect.com

[30] "Scopus Preview," 2017. [Online]. https://www.scopus.com

[31] C. Wohlin, "Guidelines for snowballing in systematic literature studies and a replication in software engineering," in *Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering*, EASE '14. ACM, 2014, pp. 38:1–38:10.

[32] M. Ivarsson and T. Gorschek, "A method for evaluating rigor and industrial relevance of technology evaluations," *Empirical Soft-*

ware Engineering, Vol. 16, No. 3, 2011, pp. 365–395.

[33] J. Cohen, "A coefficient of agreement for nominal scales," *Educational and Psychological Measurement*, Vol. 20, No. 1, 1960, pp. 37–46.

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