Technical Debt Cripples Software Developer Productivity – A longitudinal study on developers' daily software development work

Conference Paper · May 2018

DOI: 10.1145/3194164.3194178

CITATIONS
18

READS
951

3 authors:

Terese Besker
RISE Research Institutes of Sweden
27 PUBLICATIONS **355** CITATIONS
SEE PROFILE

Antonio Martini
University of Oslo
77 PUBLICATIONS **960** CITATIONS
SEE PROFILE

Jan Bosch
Chalmers University of Technology
541 PUBLICATIONS **16,192** CITATIONS
SEE PROFILE

Technical Debt Cripples Software Developer Productivity

- A longitudinal study on developers' daily software development work

Terese Besker
Computer Science and Engineering,
Software Engineering
Chalmers University of Technology
Göteborg, Sweden
Besker@chalmers.se

Antonio Martini [*] [+]
[+] CA Technologies
Strategic Research Team
Barcelona, Spain
* University of Oslo
Programming and Software Engineering
Oslo, Norway
antonima@ifi.uio.no

Jan Bosch
Computer Science and Engineering,
Software Engineering
Chalmers University of Technology
Göteborg, Sweden
Jan.Bosch@chalmers.se

ABSTRACT

Software companies need to continuously deliver customer value, both from a short- and long-term perspective. However, software development can be impeded by what has been described as Technical Debt (TD). The aim of this study is to explore the negative consequences of TD in terms of wasted software development time. This study also investigates on which additional activities this wasted time is spent and whether different types of TD impact the wasted time differently. This study also sets out to examine the benefits of tracking and communicating the amount of wasted time, both from a developer's and manager's perspective. This paper reports the results of a longitudinal study, surveying 43 software developers, together with follow-up interviews with 16 industrial software practitioners. The analysis of the reported wasted time revealed that developers waste, on average, 23% of their development time due to TD and that they are frequently forced to introduce new TD due to already existing TD. The most common activity on which additional time is spent is performing additional testing.

CCS CONCEPTS

• **Software and its engineering → Software creation and management;**

ACM Reference format:

T. Besker, A. Martini, J. Bosch, 2018. SIG Proceedings Paper in Word Format. In Proceedings of TechDebt '18, Gothenburg, Sweden, May 2018, 10 pages, https://doi.org/10.1145/3194164.3194178

TechDebt '18, May 27–28, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5713-5/18/05…$15.00
https://doi.org/10.1145/3194164.3194178

KEYWORDS

Technical Debt, Wasted Time, Software Development, Software Productivity, Longitudinal Study, Empirical Study

1 INTRODUCTION

Software companies need to continuously deliver immediate results regarding customer value, both from a short- and long-term perspective. However, software productivity can be hindered by what is described as Technical Debt (TD). TD is recognized as a critical issue in today's software development industry [28], and left unchecked, TD can lead to large cost overruns, causing high maintenance costs due to internal software quality issues [7] and inability to add new features [26] and even lead to a crisis point when a huge, costly refactoring or a replacement of the whole software needs to be undertaken [7].

The TD metaphor was introduced by Ward Cunningham [9], to describe the need to recognize the potential long-term negative effects of immature code that is made during the software development lifecycle. This debt potentially has to be repaid with interest in the long term. Interest is the negative effect in terms of the extra effort and activities that have to be paid due to the accumulated amount of TD in the system, such as executing manual processes that could potentially be automated or expending excessive effort on modifying unnecessarily complex code, or performance problems due to lower resource usage caused by an inefficient code and similar costs [28],[10].

Software suffering from TD, however, forces the developers to perform additional time-consuming activities in order to be able to continue the development work with the goal of delivering high-quality software. Sedano, Ralph and Péraire [27] echo this notion stating that *"Waste is any activity that produces no value for the customer or user"*. Hence, this wasted time negatively affects the efficiency and undermines the productivity of the developer.

There are different ways of measuring software development productivity [20]. In this study, we define productivity using the operational definition of the term productivity as the ability to deliver high-quality customer value in the shortest amount of time. This argumentation implies that a decrease in the amount of wasted software development time would lead to an increase in the software development productivity.

This study is somewhat related to a previous study [6] we previously have conducted. In that previous study, the results show that software practitioners *estimate* that, on average, waste 36% of their software development time due to experiencing TD. Even if the respondents in that study were experienced in software development, and their *estimates* were likely to be formed by what they have heard, observed, and experienced at their workplaces, we were intrigued by the idea of conducting an additional in-depth study where the wasted time could be studied by using *reported* data instead of single occurrence based on *perception* and *estimates* and also to study how those reported data varied over time.

The novelty in this study's approach compared to the previous study, lies in the selection of a longitudinal research methodology adopting a different sampling strategy, specifically focusing on developers and their *reported* experiences over a time (compared to single estimates) by collecting repetitive observations of the same variables (e.g., wasted time) on more than one single occasion and during a seven week-long period of time.

Compared to our previous study, this study had a duration of 10 months, with a longitudinal data collection phase, collecting more than 470 *reported* data from 43 software developers and a supplementary 16 follow-up interviews with both developers and their managers. Additionally, this study reveals how, frequently, developers are forced to introduce new TD caused by existing TD, and, furthermore, reports the negative effect TD has on developers' work in terms of challenges and benefits, from both developers' and managers' sides.

To the best of our knowledge, this is the first study to undertake a longitudinal study of software developers reporting their wasted time due to TD and examining which additional activities the wasted time is spent upon and also what type of TD caused the wasted time, which adds methodological novelty to the results. Furthermore, this study explores how frequently developers are forced to introduce additional TD due to already existing TD, and also investigates the awareness of the negative effect TD has on developers from both developers' and managers' perspectives.

In particular, this study will examine six main research questions:

RQ1: How much of software developers' overall development time is wasted due to Technical Debt?

RQ1.1: Is it possible to distinguish any patterns from the distribution of the wasted time over a calendar period?

RQ2: On which extra activities are the wasted time spent?

RQ3: In what ways do different Technical Debt types affect the amount of wasted time?

RQ4: How often are developers forced to introduce new Technical Debt due to already existing Technical Debt?

RQ5: How aware are the developers and their managers about the wasted time due to Technical Debt? Do developers and managers consider the insight into the wasted time useful?

This study makes a novel contribution to research, with respect to the existing body of knowledge on TD, in the following areas:

1. Based on this study's result, we show that software developers report that they waste on average *23% of their working time due to TD.*
2. We present results showing that the wasted time is most commonly spent on performing additional testing, followed by conducting additional source code analysis and performing additional refactoring.
3. The results show that in almost a quarter of all occasions when encountering TD, the developers are forced to introduce additional TD due to the already existing TD.
4. This study provides new insights into TD research by revealing that the developers are largely aware of the amount of the time they waste due to TD. However, the study shows that the developers' managers are not as aware of the amount of time developers waste, and that the different professions seem to have different views on what is a reasonable and unreasonable amount of time to waste due to TD.
5. This study shows that none of the companies tracked or measured the amount of wasted time due to TD and none of the companies had an aligned strategy for addressing the interest of TD.
6. We provide an empirically based study on how TD negatively affects practitioners within the software industry, based on both quantitative and qualitative data. A major strength of this study is the longitudinal research, which increases the validity of the results compared to cross-sectional studies.

The remainder of this paper is structured into seven sections, as follows: Section 2 introduces related work. Section 3 describes the research methods in detail. Section 4 presents the research results. Sections 5 and 6 discuss the findings and threats to the validity of the study, respectively. Finally, Section 7 concludes this study.

2 RELATED WORK

In this section, we discuss related work concerning quantifying the negative effects of TD, how TD affects the software development productivity, and contagious debt.

Ernst et al. [12] conducted a survey within three large organizations, with 536 respondents and seven follow-up interviews. Based on what the respondents in this survey stated, they found that architectural decisions are the most important source of TD. This study based their conclusion on a survey where practitioners state their *perception* of how the respondents perceive TD. In this study, we cannot find a quantification (reported or estimated) of the interest; there is no explanation regarding the types of activities on which extra-time is spent.

Holvitie, Leppanen and Hyrynsalmi [14] conducted a survey of 54 Finnish software practitioners investigating the level of TD knowledge, i.e. how TD occurs in projects. Based on the

respondents' *perceptions*, they concluded that the most frequent causes of TD were inadequate architecture and inadequate documentation.

Kazman et al. [16] present a case study for identifying and quantifying architectural debts in an industrial software project, using code changes as a proxy for calculating the interest. This study focuses on identifying the architectural roots of TD, meaning that the study does not address the cost of interest, but instead aims to quantify the expected payback for refactoring. Similar to the abovementioned related research, these costs are to some extent based on *estimated* values from the interviewed architects, and, based on these *assumptions*, the expected benefit from the refactoring is calculated.

As earlier mentioned in the Introduction section, this study is somewhat related to our previous study [6]. The previous study is based on 32 interviews and a web-survey of 258 software practitioners addressing software practitioners' *estimations* of wasted time due to TD and also the *estimations* of which types of TD have the most negative impact on daily software development work and on which different activities their respondents estimate they spend this wasted time. That study also examines how these estimations vary in relation to the age of the software and how different software roles *estimate* the variables differently. The results of that study show that software practitioners (several different roles) *estimate* that 36% of all development time is wasted due to TD and that Architectural TD and Requirement TD have the most negative impact, and that practitioners perceive that the majority of time is wasted on understanding and/or measuring the TD. In this study, we have extended our earlier study by incorporating a longitudinal study where specifically *developers report their experience* of their wasted time, their additional activities, and reported TD types (in contrast to using perceptions and estimations in the analysis).

To date, there is limited research attempting to empirically *quantifying* how TD negatively affects software development productivity. The existing literature relating to TD and productivity *states* that TD becomes a constant drain on software productivity [11], [18], which leads to a slowing down of the development and negatively affects productivity [28],[1],[4]. To the best of our knowledge, no previous study has employed a longitudinal empirical study with the aim of understanding and *quantifying* how productivity (in terms of wasted time) is affected by TD.

Our study also addresses how frequently developers are forced to introduce new TD. This topic relates to a previous study by Martini and Bosch [19], where they found that some TDs cause other parts of the system to be contaminated with the same problem, which may lead to non-linear growth of interest, called contagious debt.

3 METHODOLOGY

Triangulation is important in increasing the precision of empirical research [24]. In order to increase the validity and the reliability of the result, we have used *methodological* [21]), *source* and *observer triangulation* [24]).

3.1 Research Design

This paper is based on a longitudinal study with supplementary follow-up interviews, carried out to examine the negative impact TD has on software development, over the period of September 2016 to June 2017. This research design was divided into six phases, as visualized in Fig.1. The following sections describe each phase and the related research methods used in each stage.

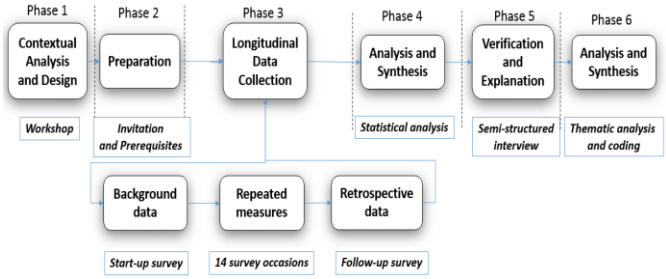

Figure 1: Visualization of the research model and research method used in each phase.

3.1.1 Contextual Analysis and Design. First, the study was presented and discussed during a workshop with software practitioners from seven software companies within our network, with an extensive range of software development. This phase acted as a guide in collecting information about the studied context and to select the most appropriate research model to use. The research team decided to base the research model on a longitudinal study together with supplementary follow-up interviews.

3.1.2 Preparation. Secondly, an invitation to participate in the study was emailed to the participants in the workshop. Following the guidelines included in [23], to those six companies (in total 43 developers) who agreed to participate in the study, we sent out educational material intended to minimize inter-observer and inter-instrument variability. All educational material and surveys have been made available at https://figshare.com/s/dfc2d9ae8e0354a4c040

3.1.2 Data Collection – Longitudinal study. A longitudinal study is a research method that involves repeated observations of the same variables (e.g., time usage) on more than one occasion [23], and is conducted over a period of time [30]. The longitudinal research method increases the precision of measuring and reduces inter-individual variation. This method also examines the individual's changing response over time and provides a value for describing both temporal changes and their dependencies on individual characteristics.

The quantitative data collection during the longitudinal study was designed and hosted by an online survey service called Survey Monkey. This phase included three different steps; the **first step** was a start-up survey gathering descriptive statistics to summarize the backgrounds of the respondents and their companies.

Based on guidelines from [17], in order to identify the population from which the subjects and objects are drawn, Table 1 presents compiled data for the participating respondents in the study.

Juristo and Moreno [15] state that "*the more homogeneous the elements examined in the surveys are, the better the results obtained will be*" and as illustrated in this table, all the respondents

were relatively experienced as software developers, where 56% have more than 10 years of experience, and only 7% had less than 2 years of experience. All respondents had had a university-level education, where 82% had a master's degree. The age of the software with which the respondents worked varied, but only 2% worked with software with an age of less than two years, and 44% of the developers worked with software within the age range of 5-10 years.

The **second step** in the longitudinal phase used *repeated measures* [23]. This stage was designed to collect reported data from 43 software developers, over 14 survey occasions (i.e. twice a week, for 7 weeks), over the period of October to November, 2016. We collected in total 473 data points, and, on average, each respondent reported their data on 11 out of 14 occasions.

In this step, we emailed out an invitation to identical online surveys to all the respondents twice a week (Tuesdays and Thursdays, with the goal of having equal spacing between the occasions, as suggested by [22]) and for those respondents who did not answer within one day, a reminder was emailed. During the entire period of this phase in the longitudinal study, the participants were asked to report their answers to the three survey questions (SQ):

- **SQ1:** How much of the overall development time have you wasted due to Technical Debt (TD), since last time you took the survey?
- **SQ2:** What extra activities was the wasted time spent on?
- **SQ3:** What was the source of the problem for which you wasted time?

In SQ1 (used for answering RQ1), the respondent reported the amount of wasted time using a value between 0 -100% of their overall working time since they last took the survey. This wording means that if the respondent for some reason did not enter the data in one or more surveys, they would enter the data from the last time the respondent took the survey. In this way, the surveys cover the full period of sampling. For question SQ2 (used for answering RQ2),, the respondent could select between the following options, for which the extra time was spent on (more than one option was selectable): "Additional code analysis", "Additional testing", "Additional communication", "Additional refactoring necessary for new implementation", "Additional searching for documentation", and "Implementing workarounds". The different listed activities were provided by [6]. In SQ3 (used for answering RQ3),, the different TD types provided by [6], were presented to distinguish the different sources on which the respondent had wasted time. The used terms were "Code-related issues", "Testing issues", "Architectural issues", "Documentation issues", "Requirement issues", and "Infrastructure issues". The respondents were asked to indicate the amount of impact each of these listed issues had on their reported wasted time using a 5-point Likert Scale (Not at All - To a Great Extent).

The final **third step** of the longitudinal data collection phase was a follow-up survey, in order to collect retrospective specific data from each respondent.

3.1.4 Analysis and Synthesis. The data collected in the fourth phase were analyzed in a quantitative fashion, i.e. by interpreting the numbers collected from the survey answers. All statistical analyses were performed with *SPSS* (version 22) and *R* version 3.3.2, using *tidyverse* [29] version 1.1.1 for data manipulation and visualization. Mixed effects models were fitted with *lme4* [5] version 1.1-12, extended with *lmerTest* version 2.0-33 for significance tests. In order to analyze the results for RQ1, RQ2, and RQ3, the proportion of the time wasted due to TD was analyzed using a linear mixed model of the form

$$\mathbf{Y^*_{ij} = \alpha + \beta_1 x_{1,ij} + \ldots \beta_p x_{p,ij} + b_i + \varepsilon_{ij}\,, \qquad (1)}$$

where the indices $i = 1,\ldots,43$ denote individual respondents and $j=1,\ldots,14$ the time-points for taking the survey. The parameters $\beta_1, \ldots, \beta_p$ are regression coefficients corresponding the explanatory variables $x_1,\ldots,x_p$, which, for example, could be activities, taking values 0 and 1, or TD types, taking values 1-5. Each individual enters the model with a subject-specific intercept $bi \sim N(0, \sigma^2_b)$, and each observation has a random error $\varepsilon_{ij} \sim N(0, \sigma^2_\varepsilon)$, which are assumed to be independent. The inclusion of random intercepts b_i accounts for inter-individual variability and intra-individual correlation. Y^*_{ij} denotes transformed proportions using the arcsine square root transformation, defined as $Y^*=(2/\pi)\sin^{-1}(\sqrt{Y})$, where Y is the proportion on the original scale. Using the arcsine transformation, the proportions were transformed into a scale where a linear model seemed plausible and approximate normality of the residuals was achieved.

The plausibility of the model assumptions was assessed by diagnostic plots and qq-plots of the residuals. Some deviations from normality were observed, primarily due to the presence of proportions equal to 0 and 1, but approximate normality was otherwise achieved. Significance tests of regression coefficients and contrasts of those were performed with approximate F-tests, using Satterthwaite's approximation of the denominator degrees of freedom [25]. Adjustments for multiple testing were performed within each group of variables (activities and TD types) using Holm's procedure [13].

3.1.5 Verification and Explanation. In the fifth phase, the results and conclusions acquired were verified using supplementary qualitative semi-structured interviews. We conducted 12 interviews with developers who had participated in the data collection phase, and four interviews with their managers. The managers were all familiar with the study but had not actively participated in the previous data collection phases. The adopted semi-structured interview technique allowed for the flexibility to explore interesting insights as they emerged. Each interview lasted between 30 and 40 minutes and was digitally recorded and transcribed verbatim. During the interviews with the developers who had participated in the data quantitative collection phase, the compiled results from their individual results were presented, and, during the interviews with their managers, an aggregated view of all the respondents from the respective company was presented. This presentation allowed the interviewees to more easily relate to the interview questions were the results of the survey were addressed.

The interview questions were primarily designed to: a) advance the understanding of the survey results, b) verify that the questions in the survey were understood as intended and in a uniform manner,

c) confirm the results of the survey, d) help to understand the implications of the results, and e) investigate how the negative effects due to TD are communicated and managed by the companies.

3.1.6 Analysis and synthesis. The transcriptions from the recorded interviews were manually coded using guidelines provided by [3]. In order to keep track of the links between the codes and the quotations, a Qualitative Data Analysis (QDA) software called Atlas.ti was used.

Based upon the research taxonomy, a coding scheme containing four broad categories and 22 individual codes was developed. Fig.2 shows the outcomes of the analysis process, where the mapping between different hierarchical categories and individual codes is graphically presented. The figure represents a reduced part of the complete data analysis model (not completely displayed here because of limitations of space).

For example, the citation *"Maybe you have to encourage the developers a bit, to get the data"* was coded as "Willingness to input data".

To ensure that the coding was performed in a consistent and reliable fashion, two authors synchronized the output of the coding, following guidelines provided by Campbell et al. [8].

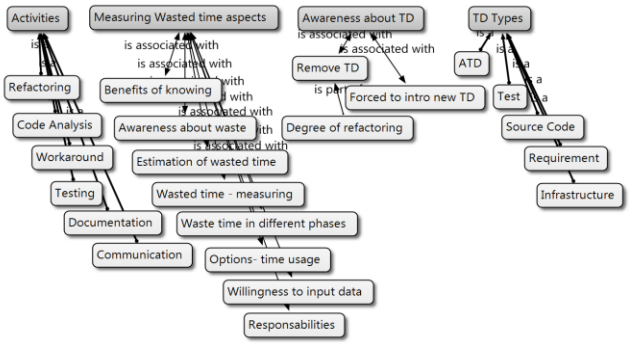

Figure 2: Sub-set of Coding Scheme.

Table 1: Characteristics of sample survey

Individual level		Company level	
Experience		**Software system type***	
< 2 years	6,98%	Embedded system	67.44%
2-5 years	16,28%	Real-time system	32.56%
5-10 years	20,93%	Data management system	11.63%
> 10 years	55,81%	System Integration	4.65%
		Modeling and/or simul.	4,65%
Educational level		Data analysis system	16.28%
Master's	81,40%	Web 2.0 / SaaS system	2.33%
Bachelor's	16,28%	Other	2.33%
No Univ. education	0,00%	**System Age**	
Other:	0,00%	< 2 years	2,33%
Ph.D.	2,33%	2-5 years	23,26%
		5-10 years	44,19%
Gender		10-20 years	25,58%
Male	83,72%	>20 years	4,65%
Female	16,28%	**Programming Language**	
		C	39,5%
		C++	20,9%
		Java	9,3%
		Python	16,3%
		Ada	7,00%
		Other	7,00%

* More than one option was selectable.

4 RESULTS AND FINDINGS

4.1 Wasted time

The first set of questions (RQ1 and RQ1.1) aim at understanding how much of software developers' overall development time is wasted due to TD and whether the distribution of the wasted time follows any patterns.

4.1.2 Wasted time (RQ1). During the longitudinal data collection phase, 43 developers reported their wasted working time due to TD twice a week for 7 weeks (in total 473 data points). On average each developer reported 10,73 times out of 14 possible occasions (with a median of 12 and std. dev. of 3,91 times) with the average time interval between the reporting occasions of 3,1 days (with a median of 2,7 and std. dev. of 1,3 days).

The single most striking observation to emerge from the data was that the respondents report that, on average, 23,1% of all software development time is wasted due to TD, with the standard deviation of 21,15% and a median value of 17,13 %. When calculating the amount of average wasted time, the different interval length between the occasions where taken into account.

Fig. 3 shows an overview of the distribution of the wasted time as a function of calendar time. It can be seen from the data in the figure that minor fluctuations of the reported wasted time are observed, but no clear trend exists.

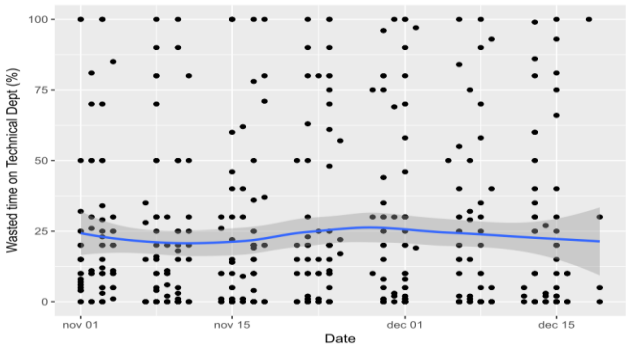

Figure 3: Wasted time on Technical Debt as a function of calendar time.

4.1.2 Distribution (RQ1.1). When examining each respondent's individual distribution of the wasted time over the study period, we noticed that variations in mean level and trend during the study period differed between the respondents, suggesting the use of a mixed effects model with subject specific intercepts, and possibly also with subject specific slopes. A better fit was, however, observed with a subject specific intercept only (see the statistical method in Section 3.1.4) and a subject specific slope was therefore not included in the analysis.

All respondents showed a different distribution of the wasted time during the study period, but when examining all different distributions, we could identify four main distribution profiles of the reported wasted time which were generally common to all of the respondents' reported data. The four identified profiles are associated with the pattern of the distribution of the wasted time

and labeled: *Fluctuating*, *Periodical*, *High*, and *Low*. Examples of these profiles are illustrated in Fig. 4. For some developers, it was evident that the wasted time varied *periodically* over time, meaning that within a sub-period of time the distribution followed a low, high or fluctuating pattern, but, at some point in time, this pattern changed. The *fluctuating* profile demonstrates that, over time, the wasted time did not show any clearly discernable time-related pattern and included both high and low amounts of wasted time. This profile was most common among all the respondents.

The *Low* and *High* profiles illustrate a wasted time that is largely consistent over time, even if some peaks can be recognized.

Finding 1: Almost a quarter of all developers' working time is reported as wasted due to having TD.

Finding 2: Even if the distribution of the wasted time varies over time for individual developers (following different identified patterns), the overall distribution of the wasted time for all developers and over time is largely consistent.

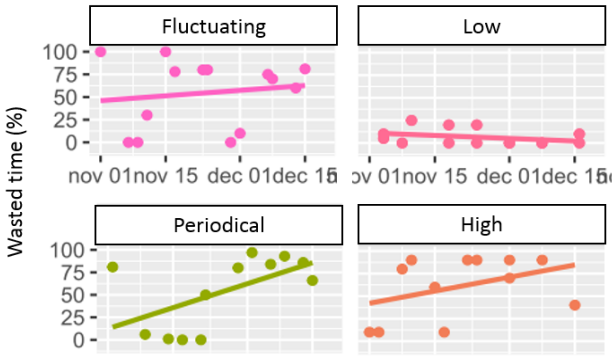

Figure 4: Different distribution profiles of the wasted time

4.2 Additional Activities

When developers encounter TD during their software development work, they are forced to perform supplementary actions in terms of performing additional activities. Accordingly, these different activities would not have been necessary if the TD was not present. The next research question (RQ2) explores the different activities on which the wasted time is spent and also if the amount of the wasted time relates to any specific activity.

During the longitudinal data collection phase, the respondents were asked to report the additional activities on which the wasted time was spent, during each occurrence (in total we collected 473 data points). For each reporting occurrence, the respondents selected the activities on which the wasted time was spent from a list of pre-defined options (listed in Section 3.1.2).

The distribution of wasted time across different activities is presented in Fig.5. The thick middle line in each box plot represents the median proportion of the wasted time per activity, and the different activities are sorted according to their strength of association with time spent on TD in multivariate analysis. For this purpose, a linear mixed effects model on arcsine square root transformed proportions was used, with subject as random effect and type of activity as fixed effects.

Figure 5: Wasted time due to Technical Debt vs. Activities.

As graphically illustrated in this figure, the activity that is most strongly associated with the wasted time is *performing additional testing*, followed by doing *additional source code analysis* and *performing additional refactoring*. The activity with the weakest association to the wasted time is *additional communication*. The "*None"* option was used when no time at all was wasted. When selecting among the different activities the wasted time was spent on, the respondents also had the possibility to enter an additional activity manually in a text field that was not pre-listen, and interestingly, no other additional activities caused by the present TD was added here. This implies that the six listed activities cover most of the extra activities caused by TD.

To evaluate if there are any statistically significant differences in the wasted time among the different activities, we made a pairwise comparison of the wasted time for the different activities. Adjustments for multiple comparisons are made with Holm's procedure [13] and adjusted p-values < 0.05 indicate a difference in waste between the activities.

The results obtained from this analysis are summarized in Table 2, where the different activities are sorted in descending order with respect to the reported wasted time. As can be seen from this Table, "Testing vs. Communication" and " Code Analysis vs. Communication " were reported as being significantly more different than the other comparisons of activities, meaning that the negative impact due to TD has a significantly different impact on these activities.

Finding 3: The activity "Performing additional testing" has the strongest association with the wasted time followed by "Additional source code analysis" and "Performing additional refactoring"

4.3 Technical Debt types

There are several different types of TD [2], including Code TD, Test TD, Architectural TD, Documentation TD, Requirement TD, and Infrastructure TD and, these different TD types could have different levels of negative impact on the amount of the wasted time. In this section, we explore in what ways these different TD types impact on wasted time and also which TD type has the most

negative impact on the wasted time from a developer's perspective. For each reporting occasion, during the longitudinal data collection phase, the respondents ranked the level of negative impact different listed TD types had on the reported wasted time, using a list of different TD types. For each listed TD type, a 5-point Likert ranking scale was set from "Not at All" to "To a Great Extent".

Table 2: Pairwise comparison of the wasted time and activities

Activities	p-value (raw)	p-value (adjusted)
Testing vs Communication	p=0.002	p=0.03
Testing vs Work-arounds	p=0.017	p=0.2
Testing vs Searching for documentation	p=0.021	p=0.23
Testing vs Refactoring	p=0.62	p=1
Testing vs Code analysis	p=0.86	p=1
Code analysis vs Communication	p=0.005	p=0.07
Code analysis vs Work-arounds	p=0.025	p=0.25
Code analysis vs Searching for documentation	p=0.032	p=0.29
Code analysis vs Refactoring	p=0.75	p=1
Refactoring vs Communication	p=0.0097	p=0.13
Refactoring vs Work-arounds	p=0.052	p=0.42
Refactoring vs Searching for documentation	p=0.062	p=0.43
Searching for documentation vs Communication	p=0.62	p=1
Searching for documentation vs Work-arounds	p=0.86	p=1
Work-arounds vs Communication	p=0.72	p=1

From the data in Table 3, it is apparent that a significant proportion of the TD encountered is related to code and less encountered TD is related to requirement and infrastructural issues.

Table 3: Likert scale of each encountered TD type

	Not at all	Very little	Little	Somewhat	To a great extent
Architectural issues	65,7%	6,6%	6,1%	17,0%	4,7%
Requirement issues	72,9%	7,8%	5,3%	11,0%	3,0%
Testing issues	64,8%	5,7%	7,6%	11,9%	10,0%
Code related issues	50,2%	6,1%	6,8%	17,6%	19,3%
Infrastructure issues	77,3%	7,0%	4,7%	6,4%	4,7%
Documentation issues	67,8%	7,0%	7,8%	11,2%	6,1%

Fig.6 illustrates each studied TD type and its relation to the reported amount of wasted time. When looking at the figure, it is apparent that the levels of encountering each of the different TD types have a positive relation to the amount of wasted time, meaning that the more of each TD type the developers encounter, the more time they waste.

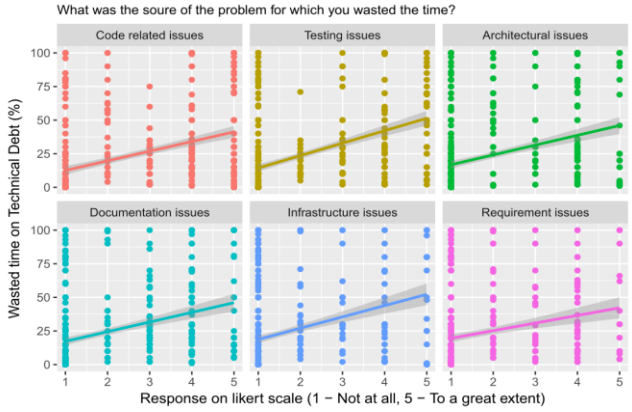

Figure 6: Wasted time on Technical Debt vs. TD types.

To assess if there was any statistical association between different TD types and the reported wasted time due to TD, both a univariable (one TD type at a time) and a multivariable (all TD types simultaneously) analysis was performed. The results from this analysis are presented in Table 4, where the different TD types are arranged in descending order (left to right, top to bottom), with respect to the strength of association with the reported wasted time. Adjustments for multiple comparisons are made with Holm's procedure [13]. Adjusted p-values < 0,05 indicate a difference in waste between TD types. What stands out in this table is that all TD types except Requirement TD are significantly related to the wasted time in both a univariable and multivariable analyses. We therefore suggest that the association of Requirement TD and the amount of wasted time is investigated in future studies.

Table 4: Association between TD type and wasted time

Model	TD-type	p-value (raw)	p-value (adjusted)
Univariable	Code related issues	p<0.001	p<0.001
Univariable	Testing issues	p<0.001	p<0.001
Univariable	Architectural issues	p<0.001	p<0.001
Univariable	Documentation issues	p<0.001	p<0.001
Univariable	Infrastructure issues	p<0.001	p<0.001
Univariable	Requirement issues	p<0.001	p=0.013
Multivariable	Code related issues	p<0.001	p<0.001
Multivariable	Testing issues	p<0.001	p<0.001
Multivariable	Architectural issues	p<0.001	p<0.001
Multivariable	Documentation issues	p<0.001	p<0.001
Multivariable	Infrastructure issues	p=0.0023	p=0.033
Multivariable	Requirement issues	p=0.69	p=1

In order to examine if there were any differences between the different TD types in relation to their negative impact on the wasted time, we used pairwise comparisons of the TD types.

In Table 5, the TD types are arranged in descending order with respect to the strength of association with wasted time. Similar to the above, adjustments for multiple comparisons are made with Holm's procedure [13], and adjusted p-values < 0,05 indicate a difference in waste between TD types. It can be seen from the data in this table that, when comparing different TD types' negative impact on the waste of time, three different comparisons return a significantly different impact on the wasted time. All these different TD types (code issues, architectural issues, and testing issues) are all related to requirement issues.

Finding 4: Aside from Requirement TD, all other TD types have a significant association between the level of negative impact and the wasted time.

4.4 Introducing new Technical Debt

Sometimes, developers are forced to introduce new additional TD due to already existing TD. This research question (RQ4) aims to address the amount of how much additional TD developers are forced to introduce due to present TD.

In the longitudinal data collection phase, the respondents reported, using a Likert scale, the level of how much additional TD they were forced to introduce because of the encountered TD.

Table 5: Pairwise comparisons of wasted time vs. TD-types.

TD-types	p-value (raw)	p-value (adjusted)
Code related vs Requirement issues	p<0.001	p<0.001
Code related vs Infrastructure issues	p=0.012	p=0.15
Code related vs Documentation issues	p=0.062	p=0.62
Code related vs Architectural issues	p=0.3	p=1
Code related vs Testing issues	p=0.81	p=1
Testing vs Requirement issues	p<0.001	p=0.0017
Testing vs Infrastructure issues	p=0.043	p=0.47
Testing vs Documentation issues	p=0.15	p=1
Testing vs Architectural issues	p=0.45	p=1
Architectural vs Requirement issues	p=0.0019	p=0.028
Architectural vs Infrastructure issues	p=0.16	p=1
Architectural vs Documentation issues	p=0.48	p=1
Documentation vs Requirement issues	p=0.023	p=0.28
Documentation vs Infrastructure issues	p=0.46	p=1
Infrastructure vs Requirement issues	p=0.069	p=0.62

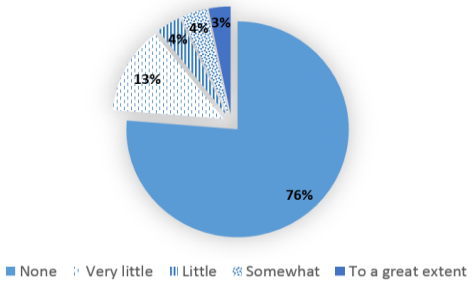

Figure 7. Introduction of new TD

The result in Fig.7 graphically illustrates that, in 24% of all the reported occasions, the developers reported that they were, to some extent, forced to introduce additional TD.

During the interviews with the developers, the majority of the interviewees explained the reasons why they were forced to introduce additional TD in terms of "time pressure".

This expressed time pressure was commonly described both in relation to the implementation of the solution, but also that it caused other activities to suffer, such as performing sufficient testing or updating-related documentations.

For example, one interviewee claimed: *"Usually, it takes a longer time to make the correct solution. It is more or less always a time question. Often when you introduce technical debt, it's because something had turned up. Which was not quite the way that we thought it was when we planned how to do the software".*

Finding 5: In a quarter of all occasions of encountering TD, developers are forced to introduce additional TD due to already existing TD, potentially causing contagious debt.

4.5 Awareness and Benefits

This research question (RQ5) focuses on the awareness of the negative consequences TD has on the daily software development work and if (and in what way) the developers and managers consider the insight of the wasted time valuable.

Aside from when participating in this study, none of the developers explicitly measured, tracked or reported their wasted time but still considered themselves to have a high level of awareness regarding the amount of time they wasted due to TD.

Initially, during the interviews with each of the developers, we asked them how much time they *estimate* they waste in general, and thereafter we showed them their individual average *reported* wasted time from the longitudinal study. When we presented the individually reported wasted time for each developer, all developers acknowledged their reported amount of time. As one interviewee stated: "*Yes, so we thought there would be some time, so it's not that we're shocked by it [22% wasted time], but it could have been worse*", while another developer commented: *"To me, it's a natural part that you waste 25%, and I think it's quite reasonable to spend so much time maintaining old code."*

On the other hand, during the interviews with the developers' managers, the general level of awareness of the amount of time developers waste due to TD was considerably lower. As one manager observed: *"As a manager, if I had data telling me that people are wasting around 25% of the time they have available for developing, I would for sure like to know that, because that's unacceptable. ... If I knew, I would be able to do something about it, or at least to raise the problem".*

These quotes also highlight the different ways developers and managers seem to appraise the amount of development time that is reasonable to waste due to TD.

Overall, both developers and managers considered the benefits of knowing the amount of wasted time in a similar way. The benefits were described by the developers as the quantified wasted time could help detecting and predicting the need for additional quality improvements. Some developers also highlighted the benefits of being able to improve forecasting and better capacity planning, and also in being able to motivate and justify it to their managers. For example, one developer stated: "*Yes, it would be useful when you do the estimation of when you start a project and when you finish it. So I could use it to predict my baselines in my delivering".* Furthermore, when discussing the quality issues, an interviewee claimed: *"If I had a huge amount of wasted time, it also shows that we have a lack of quality…. I think it would be a tool that could help us improve our quality."*

Moreover, the managers emphasize the benefits of quantifying the amount of the wasted time in terms of being able to discuss and identify various causes that adversely affect the development work. For example, one manager claimed: "*But, as a manager, I thought it was interesting because I want there to be as few barriers to my team as possible. And this is a form of obstacle. And sometimes when you ask people "what's the obstacle to you?" Then it almost becomes more an emotional question than it becomes facts."*

Finding 6: Developers have a higher awareness compared to their managers of how much time is wasted due to TD.

Finding 7: Both developers and managers described the benefits of knowing the amount of wasted time in a similar manner.

5 DISCUSSION AND LIMITATIONS

5.1 Wasted time and Introduction of new TD

The first research questions (RQ1 and RQ1.1) focus on how much software development time developers are wasting due to TD, and the fourth question (RQ4) addresses to what extent developers are forced to introduce new TD because of already existing TD.

The most striking finding shows that developers waste almost a quarter of all development time due to TD, and, even if different patterns of the distribution over calendar time were observed, the overall distribution of the wasted time did not show any clear trend. Even if this study does not explore if, and in what ways, the first introduction of TD affected the productivity of the development work, this result indicates that the present TD causes a great deal of wasted time during the overall development work. Moreover, this indicates that, if the software companies are not aware of this time and have not calculated for it, they could easily end up with time pressure, forcing them to introduce additional TD. In fact, the developers report that a quarter of all encountered TD forces them to introduce additional TD due to already existing TD. This result indicates that, depending on software companies' degree of ability to remediate TD, the amount of TD could potentially increase continuously, and, in the worst case, this could lead to a vicious cycle of TD growth. This result quantitatively corroborates the findings of [19], where the term contagious debt is described, which also supports our result.

5.2 Additional Activities

The second research question (RQ2) in this study sought to explore on which activities the wasted time was spent and also if the amount of the wasted time was related to any specific activity. The result shows that the most common activity on which the extra time was spent is performing additional *testing*, followed by additional *source code analysis* and additional *refactoring*.

This result implies that if the systems did not have TD, the time spent on these activities could be reduced. Furthermore, having to perform many of these additional activities during the software development could, consequently, be an indicator of a system suffering from TD and also an indicator of the amount of interest.

5.3 Technical Debt types

The results from the third research question (RQ3) show that, aside from Requirement issues, all other TD types are significant and strongly associated with the amount of the wasted time, whereby *Source code TD* has the strongest association on the amount of the wasted time. This result demonstrates that all different types of TD require attention. A possible explanation for these results may be that developers have a higher awareness of Source Code TD and therefore experience the negative impact caused by it to be more prominent. Likewise, the results show that developers encounter less Requirement TD and Infrastructure TD, which also point to the idea that developers are less prone to derive the wasted time to those TD types. This result further implies that software companies need to focus on several different types of TD, and not as common today, primarily focus on code related TD.

5.4 Awareness and Challenges

The fifth research question (RQ5) address the levels of awareness of the developers and their manager regarding the amount time wasted due to TD, the benefits of this insight, and how they internally communicate these issues within their organizations. From the results, we can see that software developers are reasonably aware of the amount of time they waste during the development phase, despite the fact that they do not attempt to measure, track or quantify it. However, the managers of the developers have a much lower awareness of the amount of time the developers waste and both professions also seem to have different views on what is a reasonable and unreasonable amount of time to waste on TD. If the managers are not aware of the amount of software development time the developers waste because of TD, they are consequently not able to react and take appropriate action regarding the wasted time. This means that, in a worst-case scenario, the amount of wasted time and the developer productivity could end up being increased instead of being reduced.

6 THREATS TO VALIDITY

For verifiability reasons, we have made information available online, to support a full or partial independent replication of the claimed contributions. All surveys (start-up, longitudinal, and retrospective) and the educational material used in this study are available at: https://figshare.com/s/dfc2d9ae8e0354a4c040.

There are several important limitations that necessitate a cautious interpretation of the results of the present study. The results could potentially be different in other geographical and cultural areas. Thus, further work is needed to replicate the results in other geographical areas and software development cultures. Second, given the self-reported nature of the data, the findings should be interpreted with caution, particularly because during the longitudinal data collection phase, the surveyed developers may have had insufficient knowledge and ability to categorize and quantify the correct TD type and to quantify the correct amount of wasted time due to TD. Third, a note of caution is due, since this study's result is derived from reports from *developers* and *managers* only, meaning that the findings cannot be generalized to other software practitioner roles. The result of this study may be affected by some threats to validity such as internal validity, external validity, construct validity, and reliability. The major threat to the *internal validity* of this research design is when the causal relationships between the wasted time and the different TD types and the different activities were examined, as it affects our ability to accurately explain the phenomena that we observed. To mitigate this threat, we have adopted both a univariable and a multivariable analysis of the data. The *external* aspect of validity addresses to what extent it is possible to generalize the findings [24]. Although we cannot generalize the results, we can rely on a high number of participating organizations (6). To mitigate the potential threat to the validity of self-reports, all participants reported the wasted time related to TD for a short period of time (on average 3,1 days). The confidence of self-reporting data was also supported by the fact that the practitioners knew that the surveys were coming, so they could pay special attention to their working tasks and effort spent. *Construct validity* addresses to what extent the operational measures that are studied represent what the researchers are considering and desire to investigate according to the research questions [24]. In order to mitigate this risk and to

ensure that the respondents had the same base of knowledge in the field of the study, all participants in the longitudinal study received educational material before starting the study. *Reliability* concerns to what extent the data and the analysis are dependent on the specific researchers [24]. To mitigate this threat, we have employed source, methodological, and observer triangulation.

7 CONCLUSIONS

This is the first longitudinal study of Technical Debt, where 43 developers reported, twice a week for seven weeks, how much time they waste due to TD, on what additional activities this time was spent and also what type of TD caused the wasted time. This study provides evidence that TD hinders software developers by causing an excessive amount of wasted time. This wasted time negatively affects the development productivity and viability of the software. This study shows that TD also contributes to the need for performing time-consuming additional activities, and developers report that, on average, 23% of all software development working time is wasted due to TD. Furthermore, due to the presence of TD during the development work, developers most commonly have to perform additional testing, source code analysis, and refactoring. This study also shows that, in a quarter of the occasions where developers encounter TD, they are forced to introduce additional TD due to the already existing TD. This burden of being forced to introduce additional TD demonstrates the contagiousness of TD. These findings indicate that software companies need to be armed with strategies and proactive management to enable them to track the interest of TD. Such a strategy could result in better, more informed decisions to balance the accumulation and the repayment of TD.

ACKNOWLEDGMENTS

Many thanks to the industrial partners who participated in the study. We would also like to thank Henrik Imberg for his valued support during the statistical analysis of the data. The research leading to these results has received funding from the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 712949 (TECNIOspring PLUS) and from the Agency for Business Competitiveness of the Government of Catalonia.

REFERENCES

[1] N. S. R. Alves, T. S. Mendes, M. G. de Mendonça, R. O. Spínola, F. Shull, and C. Seaman, "Identification and Management of Technical Debt: A Systematic Mapping Study," Information and Software Technology, 2015.

[2] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola, "Towards an Ontology of Terms on Technical Debt," in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, 2014, pp. 1-7.

[3] C. F. Auerbach, L. B. Silverstein, and Ebrary, Qualitative data: an introduction to coding and analysis. New York: New York University Press, 2003.

[4] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, "Reducing friction in software development," IEEE Software, vol. 33, no. 1, 2016, pp. 66-72.

[5] D. Bates, M. Maechler, B. Bolker, and S. Walker, "Fitting Linear Mixed-Effects Models Using lme4.," Journal of Statistical Software, no. 67(1), 2015, pp. 1-48.

[6] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt - When and by whom will it be paid? ," in IEEE International Conference on Software Maintenance and Evolution (ICSME), Shanghai, China, 2017, pp. 13-23.

[7] T. Besker, A. Martini, and J. Bosch, "Time to Pay Up - Technical Debt from a Software Quality Perspective," in proceedings of the 20th Ibero American Conference on Software Engineering (CibSE) @ ICSE17, Buenos Aires, Argentina, 2017, p. pp. in print. .

[8] J. L. Campbell, C. Quincy, J. Osserman, and O. K. Pedersen, "Coding In-depth Semistructured Interviews Problems of Unitization and Intercoder Reliability and Agreement," Sociological Methods & Research, 2013.

[9] W. Cunningham, "The WyCash portfolio management system, in: 7th International Conference on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA '92)," 1992, pp. 29–30.

[10] B. Curtis, J. Sappidi, and A. Szynkarski, "Estimating the size, cost, and types of technical debt," presented at the Proceedings of the Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012.

[11] R. J. Eisenberg, "A threshold based approach to technical debt," SIGSOFT Softw. Eng. Notes, vol. 37, no. 2, 2012, pp. 1-6.

[12] N. A. Ernst, S. Bellomo, I. Ozkaya, R. L. Nord, and I. Gorton, "Measure it? Manage it? Ignore it? software practitioners and technical debt," presented at the Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 2015.

[13] S. Holm, "A Simple Sequentially Rejective Multiple Test Procedure," Scandinavian Journal of Statistics, vol. 6, 1979, pp. 65-70.

[14] J. Holvitie, V. Leppanen, and S. Hyrynsalmi, "Technical Debt and the Effect of Agile Software Development Practices on It - An Industry Practitioner Survey," in Managing Technical Debt (MTD), 2014 Sixth International Workshop on, 2014, pp. 35-42.

[15] N. Juristo, A. M. Moreno, SpringerLink, and A. SpringerLink, Basics of Software Engineering Experimentation, 1 ed. Boston, MA: Springer US, 2001.

[16] R. Kazman, C. Yuanfang, M. Ran, F. Qiong, X. Lu, S. Haziyev, *et al.*, "A Case Study in Locating the Architectural Roots of Technical Debt," in Software Engineering (ICSE), 2015 IEEE/ACM 37th IEEE International Conference on, 2015, pp. 179-188.

[17] B. A. Kitchenham, S. L. Pfleeger, L. M. Pickard, P. W. Jones, D. C. Hoaglin, K. El Emam, *et al.*, "Preliminary guidelines for empirical research in software engineering," Software Engineering, IEEE Transactions on, vol. 28, no. 8, 2002, pp. 721-734.

[18] 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.

[19] A. Martini and J. Bosch, "On the interest of architectural technical debt: Uncovering the contagious debt phenomenon," Journal of Software: Evolution and Process, 2017.

[20] K. D. Maxwell, "Software Development Productivity," Advances in Computers, vol. 58, 2003/01/01/, 2003, pp. 1-46.

[21] J. Miller, "Triangulation as a basis for knowledge discovery in software engineering," Empirical Software Engineering, vol. 13, no. 2, 2008, pp. 223-228.

[22] D. F. Morrison, "The optimal spacing of repeated measurements," Biometrics, vol. 26:281-90, 1970.

[23] R. E. Ployhart and R. J. Vandenberg, "Longitudinal Research: The Theory, Design, and Analysis of Change," Journal of Management, vol. 36, no. 1, 2010/01/01, 2009, pp. 94-120.

[24] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," Empirical Software Engineering, vol. 14, no. 2, 2009, pp. 131-164.

[25] G. B. Schaalje, J. B. McBride, and G. W. Fellingham, "Adequacy of Approximations to Distributions of Test Statistics in Complex Mixed Linear Models," Journal of Agricultural, Biological, and Environmental Statistics, vol. 7, no. 4, 2002, pp. 512-524.

[26] C. Seaman, G. Yuepu, N. Zazworka, F. Shull, C. Izurieta, C. Yuanfang, *et al.*, "Using technical debt data in decision making: Potential decision approaches," in Managing Technical Debt (MTD), 2012 Third International Workshop on, 2012, pp. 45-48.

[27] T. Sedano, P. Ralph, and C. e. Péraire, "Software development waste," presented at the Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, Argentina, 2017.

[28] 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.

[29] H. Wickham, "tidyverse: Easily Install and Load 'Tidyverse' Packages. R package version 1.1.1," in URL https://CRAN.R-project.org/package=tidyverse, 2017.

[30] C. Wohlin, P. Runeson, M. Höst, M. C. Ohlsson, B. Regnell, and A. Wesslen, Experimentation in software engineering: an introduction: Kluwer Academic Publishers, 2000.