Software Developer Productivity Loss Due to Technical Debt - A replication and extension study examining developers' development work

Terese Besker , Antonio Martini , Jan Bosch

PII: S0164-1212(19)30133-5
DOI: https://doi.org/10.1016/j.jss.2019.06.004
Reference: JSS 10366

To appear in: *The Journal of Systems & Software*

Received date: 10 January 2019
Revised date: 23 May 2019
Accepted date: 5 June 2019

Please cite this article as: Terese Besker , Antonio Martini , Jan Bosch , Software Developer Productivity Loss Due to Technical Debt - A replication and extension study examining developers' development work, *The Journal of Systems & Software* (2019), doi: https://doi.org/10.1016/j.jss.2019.06.004

Highlights

- Almost a quarter of all developers' working time is reported as wasted
- Additional code analysis has the strongest association with wasted time
- A quarter of all cases of Technical Debt, forces developers to introduce new Debt
- Developers have higher awareness than managers of how much time is wasted

Software Developer Productivity Loss Due to Technical Debt

- A replication and extension study examining developers' development work

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

Antonio Martini *+
* 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 deliver customer value continuously, both from a short- and long-term perspective. However, software development can be impeded by technical debt (TD). Although significant theoretical work has been undertaken to describe the negative effects of TD, little empirical evidence exists on how much wasted time and additional activities TD causes. The study aims to explore the consequences of TD in terms of wastage of development time. This study investigates on which activities this wasted time is spent and whether different TD types impact the wasted time differently. This study reports the results of a longitudinal study surveying 43 developers and including16 interviews followed by validation by an additional study using a different and independent dataset and focused on replicating the findings addressing the findings. The analysis of the reported wasted time revealed that developers waste, on average, 23% of their time due to TD and that developers are frequently forced to introduce new TD. The most common activity on which additional time is spent is performing additional testing. The study provides evidence that TD hinders developers by causing an excessive waste of working time, where the wasted time negatively affects productivity.

Keywords—Software Development, Software Productivity, Technical Debt, Wasted Development Time

1 INTRODUCTION

To survive in today's fast-growing and ever-changing business environments, large-scale software companies need to deliver customer value continuously, from both a short- and long-term perspective.

During the software development lifecycle, companies need to consider the costs of the software development process in terms of the required time and resources. In general, software companies strive to increase the number of implemented features, the overall software quality, and the overall efficiency and, at the same time, decrease the costs in each lifecycle phase by reducing time and resources deployed by the development teams.

However, software development 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 [1] and, left unchecked in the software, TD can lead to large cost overruns, causing high maintenance costs due to internal software quality issues [2],[3],[4],[5],[6],[7],[8] and an inability to add new features [9] and even lead to a crisis point when a huge, costly refactoring or a replacement of the whole software needs to be undertaken [10].

The TD metaphor was first coined at OOPSLA '92 by Ward Cunningham [11] to describe the need to recognize the potential long-term negative effects of immature code that occur during the software development lifecycle. Cunningham used the financial terms debt and interest when describing the concept of TD:

> "*Shipping first-time code is like going into debt. A little debt speeds development so long as it is paid back promptly with a rewrite. Objects make the cost of this transaction tolerable. The danger occurs when the debt is not repaid. Every minute spent on not-quite-right code counts as interest on that debt.*"

An additional, and more recent, definition was provided by Avgeriou et al. [12] who define TD as: *"In software-intensive systems, technical debt is a collection of design or implementation constructs that are expedient in the short term, but set up a technical context that can make future changes more costly or impossible. Technical debt presents an actual or contingent liability whose impact is limited to internal system qualities, primarily maintainability and evolvability."*

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 software. This may include 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 [1],[13].

Software suffering from TD, however, forces the developers to perform additional time-consuming activities to be able to continue the development work with the goal of delivering high-quality software. Accordingly, an extensive amount of valuable developing working time is wasted when developers are forced to execute these additional activities due to TD in their software systems, and thus this wasted time negatively affects the efficiency and undermines the productivity of software developers. This study aims at increasing the understanding of the negative effects of experiencing TD by using weekly reporting of the wasted time over time since it is essential to have informative and cost-effective indicators to evaluate aspects of the

software development processes and its product quality [14].

There are different ways of measuring software development productivity [15], and productivity is typically defined as the output divided by the effort required to produce the output. Following guidelines by Fonseca [16], we specify the operational definition [17] of developer productivity by examining the developers' reported amount of wasted time using weekly web surveys over seven weeks. Since we can determine that the amount of wasted time has a negative impact on software development duration, we define productivity as *the ability to deliver high-quality customer value in the shortest amount of time*. This line of reasoning implies that a decrease in the amount of wasted software development time would lead to an increase in software development productivity.

Examining and quantifying the negative effects of TD plays an important role for both academia and software management practitioners in understanding and raising awareness of the magnitude of the negative effects TD has on developers' productivity. This knowledge can also help to improve software development efficiency and strategies for TD management.

The results of this study show that TD has a negative effect in terms of an extensive amount of developer working time (on average 23%) wasted due to experiencing TD during the software development lifecycle. This study also demonstrates the variance of the wasted time across the sample and, furthermore, that 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.

To the best of our knowledge, this is the first study to undertake a longitudinal examination of software developers reporting their wasted time due to TD and examining which additional activities the wasted time is spent on and also what type of TD caused the wasted time, which also adds methodological novelty to the results.

The remainder of this study is structured in seven sections: Section 2 describes the research questions, and Section 3 introduces related work. Section 4 describes the research methods in detail. Section 5 presents the research results. Sections 6 and 7 discuss the findings and threats to the validity of the study, respectively. Finally, in Section 8, conclusions and recommendations for future work will be presented.

This manuscript was originally published at the 1st International Conference on Technical Debt, held jointly with ICSE [18]. The delta of this manuscript over the prior published paper is two-sided. First this manuscript is an *extension study* of a previous paper, and secondly, this study is a *replication study* of the original study.

This new manuscript includes a value-added extension to the previous conference version of the study since that version was restricted due to space limitations. This manuscript has been extended to include additional research questions and, consequently, new findings to these research questions. The related Research section has been extended to be broader and more carefully cover additional related research publications. In the Methodology section, the novel approach of using a longitudinal research method is described in greater depth. In the Results section, we have added more subjects and more rigorous analysis by including more examples and

explanations, which may increase our confidence in the conclusions. In this extended version of the previous paper, several of the figures and tables have also been refined to strengthen further the readability and understandability of the results. This manuscript also includes a validation by replication of the findings focusing on the amount of wasted time and the different encountered TD types and which additional activities this time is spent on. This replication phase confirms and strengthens the results derived in the original study. Furthermore, this manuscript includes additional in-depth analysis of how the results can be applied in practice, both by practitioners and by researchers. Finally, this manuscript also includes a potential direction for future work.

2 Research Questions

The goal of this study, phrased as inspired by the goal-question-metric approach provided by Basili [19] is: "To *analyze* the consequences of TD, *for the purpose of* understanding, *with respect to* the negative effect TD has on software productivity, *from the point of view* of software developers and their managers, *in the context of* software development." Based on this goal and more specifically, this study will examine the following six research questions:

RQ1: In what ways does technical debt waste developers' working time?

RQ1.1: How much of software developers' overall development time is wasted due to technical debt?

RQ1.2: Is there a significant difference between the wastage of working time due to technical debt in relation to different developer characteristics?
In this regard, the characteristics refer to different variables such as years of experience as a developer, gender, level of education, programming language, company, the age of the software, and type of software.

RQ1.3: Are there different patterns among the distributions of the wasted time over the calendar period?
The objectives of research question RQ1 (RQ1.1, RQ1.2, and RQ1.3) are to understand how much of software developers' overall development time is wasted due to TD and whether the distribution of the wasted time varies with developer characteristics and follows any specific distribution patterns.

RQ2: Upon which extra activities is 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: Is there a difference in the awareness of technical debt between developers and their managers?
The objective of this research question 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 into the wasted time valuable.

RQ6: What are the challenges in tracking the interest of technical debt?

3 Related Work

In this section, we discuss related work concerning software productivity, the quantification of negative effects due to TD, how TD affects the software development productivity, and, finally, the term contagious debt.
Software productivity has been a frequently discussed subject since the beginning of software engineering research [20],[21]. In software engineering, productivity is commonly defined, from an economic point of view, as the effectiveness of productive effort measured in terms of the rate of output per unit input [21],[20],[22].
Productivity is also a measure of the quality of an output relative to the input required to produce the output. Productivity is a combined measurement of efficiency and quality. There are several constraints that can influence the software development productivity in general, such as cost, schedule, and scope [23]. Furthermore, Oliveira et al. [20] state that researchers have not yet reached a consensus on how to measure productivity properly in software engineering. However, as mentioned in the Introduction, in this study we have decided to focus on the productivity in terms of the amount of *wasted* software development time because developers are hindered during their daily software development work by experiencing TD within their software.
Sedano, Ralph, and Péraire [24] echo this notion by stating that *"waste is any activity that produces no value for the customer or user"* and reducing this waste of time would improve the software development productivity. In their study, they identified that software suffering from TD can lower the developer's productivity both in terms of wasted time and by causing reworking and an extraneous cognitive load.
Ernst et al. [25] surveyed three large organizations with 536 respondents and seven follow-up interviews. Based on the responses from this survey, they found that architectural decisions are the most important source of TD. This study based its conclusion on a survey in which 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 also no explanation regarding the types of activities on which the extra time is spent.
Martini and Bosch [26] have studied the interest growth and found that some Architectural TD items are contagious, causing the interest to be not only fixed but potentially compounded, which leads to the hidden growth of interest with the potentiality of growing exponentially. Even if that study included some cases, the study did not track to the introduction of additional

TD using the same level of granularity as this study.
Kazman et al. [27] 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. However, besides estimates, that study also used revision history data to calculate the development cost.
This study is, to a certain extent, related to our previous study [28], which also addresses the amount of wasted software development time. That previous study was 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 the respondents estimate they spend this wasted time. The results of that study show that software practitioners (from several different roles) *estimate* that, on average, 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.

The motivation for conducting this study, was to base the results on repeated reported data and to specifically target developers and thereby gain a better and deeper understanding of the waste of time caused by TD, by collecting more data points and to combine the study with a replication of the results in order to validate whether the findings can be repeatedly generated.
Later on, when studying the time spent managing TD, we found, in a study by Martini et al. [29], that the development time dedicated to managing TD is *estimated* to be, on average, 25% of the overall development time and generally not performed systematically
The novelty of this study's approach compared to the previous studies lies in the selection of a longitudinal research methodology adopting a different sampling strategy specifically focusing on developers and their *reported* experiences over time (compared to single estimates) by collecting repetitive observations of the same variables (e.g., wasted time) on more than one single occasion [30] and over seven weeks. The uniqueness of this extended study is also that this study validates several of the findings by conducting an additional replication study using a new and independent data set.
Compared to previous studies, this original study had a duration of 10 months, with a longitudinal data collection phase, collecting more than 470 *reported* data from 43 software developers and 16 supplementary follow-up interviews with both developers and their managers, in comparison with the previous studies [28] and [29], where the results were based on single estimates from the participants. Additionally, this study also reveals new insights and interpretations on how, developers are frequently 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 date, there is limited research available that attempts to quantify empirically 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 [31],[32], which leads to a slowing of the development and negatively affects productivity [1],[33],[34]. To the best of our knowledge, no previous study has employed a longitudinal empirical study (based on reported data) with the aim of understanding and quantifying how productivity (in terms of wasted software development 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 [26] in which they found that some TDs cause other parts of the system to be contaminated with the same problem, which may lead to the non-linear growth of interest, called contagious debt.

4 Methodology

Triangulation is important in increasing the precision of empirical research in terms of taking different perspectives toward the studied object and thus providing a broader view [35]. To increase the validity and the reliability of the result, we have used source, observer, and methodological triangulation and also replicated the study as a last phase.

Source triangulation refers to using several sampling strategies to ensure that data are gathered at different times and in different situations. The use of more than one source of data makes the conclusion more credible since it can be drawn from several sources of information [35], [36]. In this study, we used both interviews and surveys when collecting the data.

Observer triangulation refers to using more than one observer to gather and interpret data [35], [36]. This type of triangulation was achieved in this study, where at least two of the involved researchers worked together with different roles during the studies, thus enabling peer debriefing and analysis of the collected data.

Methodological triangulation refers to combining different types of data collection methods and was achieved in this study by combining both qualitative and quantitative methods [35], [36].

4.1 Research Design

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

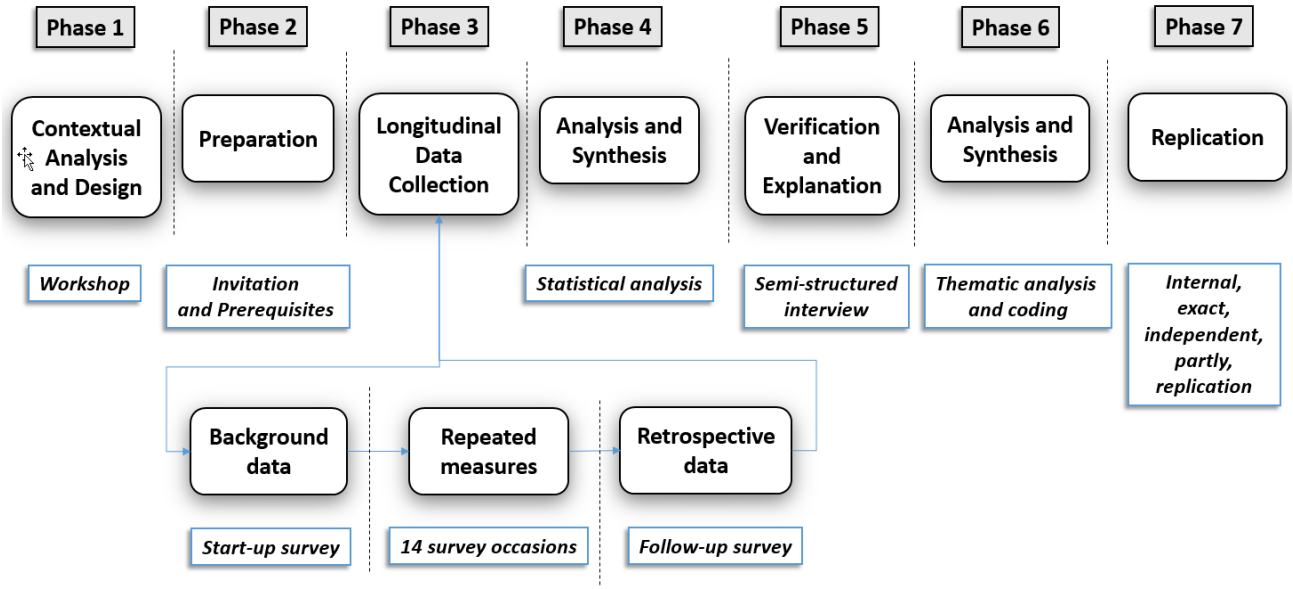


Fig. 1. Visualization of the research design and research method used in each phase

4.1.1 Contextual Analysis and Design. First, the study was presented and discussed during a workshop with software practitioners from seven software companies within our network and with an extensive range of software development. This study's selection of participating companies was carried out with a representative convenience sample of software professionals from our industrial partners.

This phase acted as a guide for collecting information about the studied context and selecting 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.

4.1.2 Preparation. Secondly, an invitation to participate in the study was emailed to the participants in the workshop. Following the guidelines provided by Ployhart and Vandenberg [30], to those six companies (43 developers in total) who agreed to participate in the study, we sent out educational material intended to minimize inter-observer (all researchers communicate the same knowledge) and inter-instrument (all participants receive the same information) variability.

This educational material included different topics such as TD definitions, different terms commonly related to TD (e.g., debt, principal, and interest), TD Landscape (also illustrating what TD is not), and, finally, a short description of the different TD types. Since the definition of TD is very important and sets the context for the entire study, we specifically focused the educational material on guiding the respondents to fully understand the basic concepts of TD. In the material, we, therefore, presented three different citations to describe what is meant by TD. First, Ward Cunningham's definition was [11] offered: *"Shipping first time code is like going into debt. A little debt speeds development so long 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 the debt," followed by Steve McConnell's definition of TD [37]:*"A design or construction ap-*

proach that's expedient in the short term but creates a technical context in which the same work will cost more to do later than it would cost to do now (including increased cost over time)." Furthermore, a third, shorter, definition was used: "*Technical debt is a non-optimal solution in code (or other artifacts related to software development) that gives a short-term benefit but cause an extra long-term cost during the software life-cycle."*

4.1.3 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 [30], and is conducted over time [38].

The incentive for using a longitudinal research method in this study has two principal aspects:

a) to increase the precision of reporting experienced data (in our case, not based on single estimations and single perceptions). This was achieved by studying each respondent over several weeks where the reported data could be compared. Such designs are called repeated measures designs [30], and

b) to examine the respondents' changing responses over time: Longitudinal designs have a natural appeal for the study of changes associated with development or changes over time. They have value for describing both temporal changes and their dependence on individual characteristics [30]. The longitudinal research method increases the precision of measuring and reduces inter-individual variation. This method also examines the individual's changing responses over time and provides a value for describing both temporal changes and their dependencies on individual characteristics [30].

To sustain the commitment of the respondents, before starting the study, all respondents had agreed on their continuing participation with both their managers and ourselves.

The quantitative data collection during the longitudinal study was designed and hosted by an online survey service called Survey Monkey. This data collection 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 [39] and to identify the population from which the subjects and objects are drawn, we studied compiled data for the participating respondents in the study. Juristo and Moreno [40] state that "*the more homogeneous the elements examined in the surveys are, the better the results obtained will be*," and, as illustrated in Table 1, all the respondents were relatively experienced as software developers: 56% had more than 10 years of experience, and only 7% had fewer than 2 years of experience. All respondents have a university-level education, where 82% have 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 2 years, and 44% of the developers worked with software within the age range of 5-10 years. The most common system type was an embedded system, and the most common programming languages were C (40%) and C++ (21%).

The second step in the longitudinal phase used repeated measures [30]. This stage was designed to collect reported data from 43 software developers more than 14 survey occasions (i.e., twice a week for seven weeks) from October to November 2016.

Out of 602 possible data collections (43 developers multiplied by 14 occasions), we collected 473 data points. However, since we asked the respondents to report their data experiences since the last time they took the survey, also these "missing" reporting occasions were covered in the data collection. On average, each respondent reported their data on 11 out of 14 occasions.
In this step, we emailed out an invitation to an online survey to all the respondents, twice a week (Tuesdays and Thursdays), with the goal of having equal spacing between the occasions, as suggested by Morrison [41], 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), and all reported data from the survey were used in the study:

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

In survey question SQ1 (used for answering RQ1), the respondents reported the amount of wasted time using a value between 0-100% of their overall working time since they last took the survey. To address the potential problem with missing data from the respondents, if, for some reason, the respondents did not enter the data in one or more surveys, the respondents were always asked to report their experienced data *since the last time* they took the survey. This wording means that if, for some reason, the respondent 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 survey question SQ2 (used for answering RQ2), the respondent could select between the following options on which the extra time was spent (more than one option was selectable): "Additional code analysis", "Additional testing", "Additional communication", "Additional re-factoring necessary for new implementation", "Additional searching for documentation", "Implementing workarounds" and "other". If ticking the "other" option, the respondents were asked to textually describe this activity in a comment field. The listed activities were provided by Besker et al. [28].
In survey question SQ3 (used for answering RQ3), the different TD types provided by Besker et al. [28] were presented to distinguish the different sources on which the respondent had wasted time. The terms used were "Code-related issues," "Testing issues," "Architectural issues," "Documentation issues," "Requirement issues," "Infrastructure issues," and "Other." If ticking the "Other" option, the respondents were asked to textually describe this issue in a comment field.
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 to collect

retrospective specific data from each respondent.

4.1.4 Analysis and Synthesis. In the fourth phase, the data collected were analyzed qualitatively, that is, by statistical analysis of the data collected from the survey answers.
For descriptive purposes, data is summarized by mean, median, and standard deviation for continuous variables, and numbers and percentages for categorical variables. The distribution of waste by the developer and by their characteristics is presented and analyzed graphically using boxplots. The strength of association between waste and various explanatory variables, such as developer characteristics and TD-types, is summarized by R-squared, describing the fraction of variance in wasted time explained by a set of explanatory variables.
Multivariable analyses of waste related to developer characteristics were also performed, aiming at finding combinations of developer characteristics associated with greater or lower waste. For this purpose, non-parametric regression trees were used. Model evaluation was performed by leave-one-out cross-validation.
The distribution of waste over time, overall, and by subject was analyzed graphically by scatterplots overlaid by smooth regression curves estimated using LOESS.
Analyses of longitudinal data were performed using logit-linear mixed effects models with the fraction of wasted time as the response variable. Activities or TD types were included as categorical explanatory variables in the models, and subject-specific intercepts and time effects were included as random effects. Random effects were included to account for subject-specific time trends and intra-individual correlations. Robust standard errors of the parameter estimates were used to obtain standard errors and confidence intervals that are robust against heteroscedasticity. Modeling was performed on a logit scale, motivated by the bounded nature of the response variable, and the effect of TD activities on waste on the original scale was computed as follows. The average effect on waste was computed by artificially changing the activity to inactive or active state for each data record, accounting for both subject effects and other concurrent activities. A confidence interval for this statistic was computed using the non-parametric bootstrap percentile method with 10000 bootstrap replicates.
Statistical analyses were performed with SPSS (version 22), SAS 9.4 (SAS Institute, Cary NC) using the GLIMMIX procedure for logit-linear mixed effects models, and R version 3.4.3 [42] using the rpart package version 4.1-13 for non-parametric regression trees [43].

4.1.5 Verification and Explanation. In the fifth phase, the results and conclusions acquired in previous phases were verified using supplementary qualitative semi-structured interviews. We conducted 12 interviews with developers and four interviews with their managers. All the developers had participated in the previous data collection phases, and the managers were all familiar with the study but had not actively participated in the previous data collection phases.
As suggested by Seaman [44], this study employed semi-structured interviews including a mixture of open-ended and specific questions designed to elicit not only the information foreseen but also unexpected types of information. In the interviews, the questions were planned but not necessarily asked in the same order as they were listed. This interview technique allowed for the flexibility to explore interesting insights as they emerged.

Each interview lasted between 30 and 40 minutes, and, to obtain a more accurate rendition of the interviews, all interviews were digitally recorded and transcribed verbatim. All interviewees were asked for recording permission before starting, and they all agreed to be recorded and to be anonymously quoted for this paper.
During the interviews with the developers who had participated in the quantitative data 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 where the results of the survey were addressed. Some interview questions had a focus on corroborating certain findings that we already thought had been established during previous data collection activities, where the questions were carefully worded (avoiding leading questions) to allow the interviewee to provide fresh commentaries on them [45].
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.

4.1.6 Analysis and synthesis.

When analyzing the qualitative data collected in this thesis, a thematic analysis approach was used. Thematic analysis is a method for identifying, analyzing, and reporting patterns and themes within data, which involves searching across a dataset to find repeated patterns of meaning. The thematic analysis provides a flexible and useful research tool, which offers a detailed and complex explanation of the collected data [46].
When analyzing the qualitative data, the guidelines provided by Braun and Clarke [46] were used to conduct the analysis in a thorough and rigorous manner. The thematic analysis was conducted using a six-phase guide.
First, the audio-recorded qualitative data collected from interviews was transcribed into written form, where we were also able to familiarize ourselves with the data. The second step involved the production of initial codes from the data, where we organized the data into meaningful groups. In this phase of the analysis, a qualitative data analysis (QDA) software package called Atlas.ti was used. The third phase focused on searching for themes by sorting the different codes into potential themes and collating all the relevant coded data extracts within each identified theme. Each extract of data was assigned to at least one theme and, in many cases, to multiple themes. For example, the citation *"Maybe you have to encourage the developers a bit, to get the data*" was coded as "*Willingness to input data*" in the theme "*Measuring Wasted Time Aspects.*" To ensure that the coding was performed in a consistent and reliable fashion, triangulate the interpretation of the data, and avoid bias as much as possible, two authors synchronized the output of the coding, following guidelines provided by Campbell et al. [47].
The fourth phase focused on the revised set of candidate themes involving the refinement of those themes. The refinement focused on forming coherent patterns within the themes. Otherwise, we revised the themes or created a new theme. The fifth phase focused on identify-

ing the essence of each theme and determining what aspect of the data each theme captured. This phase also stressed the importance of not just paraphrasing the content of the data extracts, but also identifying what is interesting about them and why. The final phase of the thematic analysis took place when we had a set of fully developed themes and involved the final analysis.

Based on the research taxonomy, a coding scheme containing four broad themes and 22 individual codes was developed. Fig. 2 shows the outcome of the analysis process, where the mapping between different hierarchical categories and individual codes are graphically presented. The categorized data collected during the interviews was used during the analysis of the study and are reported in Section 5.

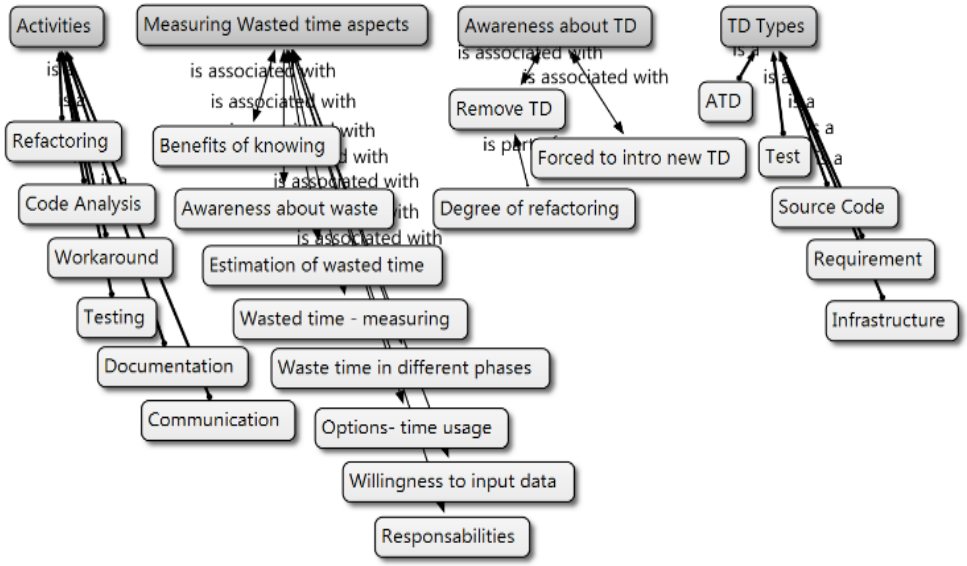


Fig. 2. Coding Scheme

TABLE 1
CHARACTERISTICS OF RESPONDENTS

Individual-level	No. of respondents	(%)	Company level	No. of respondents	(%)
Experience			**Software system type***		
< 2 years	3	(6.98%)	Embedded system	29	(67.44%)
2-5 years	7	(16.28%)	Real-time system	14	(32.56%)
5-10 years	9	(20.93%)	Data management system	5	(11.63%)
> 10 years	24	(55.81%)	System Integration	2	(4.65%)
			Modeling and/or simul.	2	(4.65%)
Educational level			Data analysis system	7	(16.28%)
Master's	35	(81.40%)	Web 2.0 / SaaS system	1	(2.33%)
Bachelor's	7	(16.28%)	Other	1	(2.33%)
No. Univ. education	0	(0.00%)	**System Age**		
Other:	0	(0.00%)	< 2 years	1	(2.33%)
Ph.D.	1	(2.33%)	2-5 years	10	(23.26%)
			5-10 years	19	(44.19%)
Gender			10-20 years	11	(25.58%)
Male	36	(83.72%)	>20 years	2	(4.65%)
Female	7	(16.28%)	**Programming Language**		
			C	17	(39.5%)
			C++	9	(20.9%)
			Java	4	(9.3%)
			Python	7	(16.28%)
			Ada	3	(6.98%)
			Other	3	(6.98%)

* More than one option was selectable.

4.1.7 Internal Exact Independent Partly Replication.

Replication plays a key role in empirical software engineering [48], and is proposed as an important means of increasing confidence and assessing reliability in the result [49], [50].

As illustrated in Fig. 1, the original study consists of six different research phases, followed by a seventh phase in the replicated study. After the first sixth research phases, we were able to answer all stated research questions. The replicated study aimed at answering three of the identical research question used in the original study. After the results from the replicated study were derived, these results were compared with the results from the original study to either confirm or disconfirm the original findings.

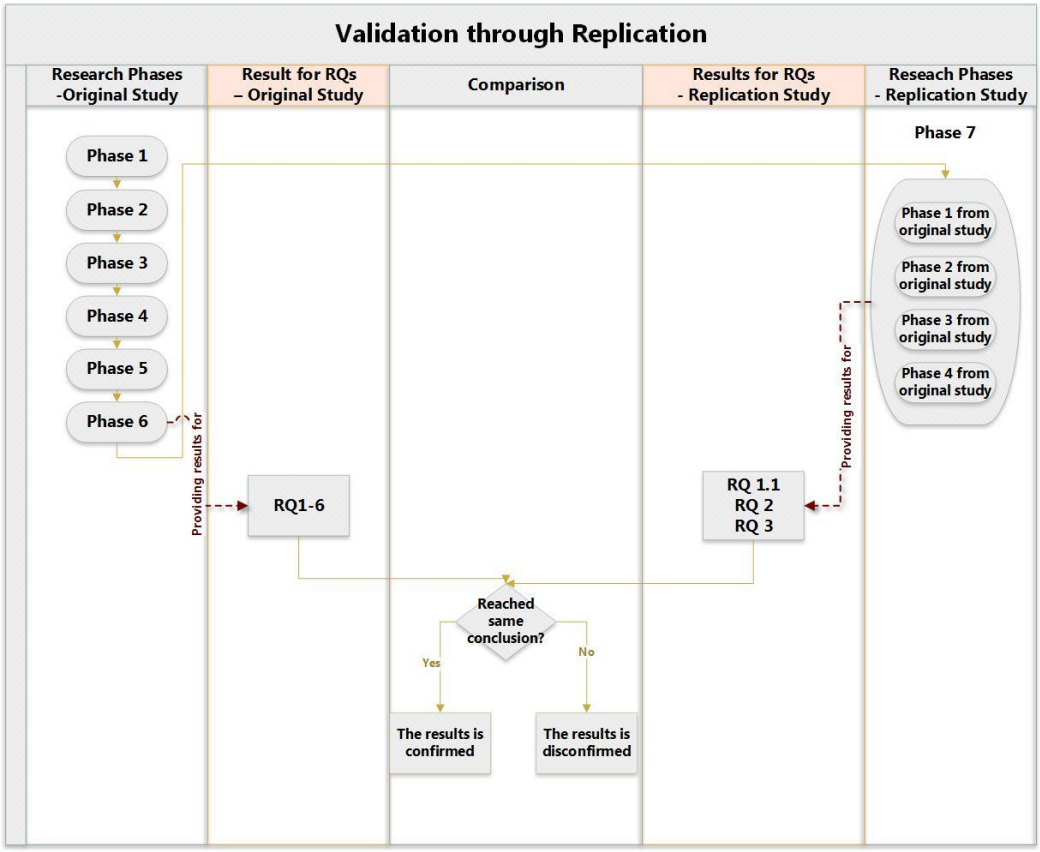


Fig. 3. Integration of Replication study

As described above, in the seventh phase in this study we have conducted an *internal exact independent partly replication* of the study, where the *Internal* reflects that the replication team is the same as in the previous phases. The categorization *Exact* reflects that the procedures are followed as closely as possible to determine whether the same results can be obtained. To gain more insight into the original results, this categorization also includes replications that are modified to some extent by, for example, altering the subject pool or other conditions. The categorization of *Independence* reflects that, during the replication phase of the study, we deliberately varied some aspects of the conditions when collecting the data [48]. The categorization *Partly* points out that not all of the research areas and research questions are replicated. More specifically, this phase aims at replicating the findings for RQ1.1, RQ2, and RQ3 addressing the overall amount of wasted time due to TD and also the extent of encountering different types of TD.

Due to restrictions on how this replication study could be carried out by the involved company, and what data was feasible to collect, the datasets in this replication study do not include enough information to allow for replications of all original research questions. For instance, the replication study did not include any follow-up interviews with the participating developers and their managers. This limitation resulted in RQ5 and RQ6 not being replicated in this phase of the study, and they were, therefore, omitted.

The design of this phase follows the guidelines proposed by Carver [51] for reporting replication studies. The motivation for conducting this replication phase was to validate the results from the original study by changing the participant pool and the way we collected data to gain additional confidence that the original results were not the result of, for example, a data collec-

tion bias or the selection of the study design. Below, we describe each activity for the replication study, using the same phases as we used in the original study, to illustrate similarities and/or differences between the two sets of studies.

In the first phase (in comparison with section 4.1.1) of the replication study, this part of the study was presented to *one* specific software company in Germany, with an extensive range of software development. The company name has been anonymized for confidentiality reasons. The research team decided with the contact persons at the company on a suitable research model to use in this replication phase of the study.

During a second phase (in comparison with section 4.1.2) in the replication phase, the contact persons in the company invited developers to voluntary participation in the study. Similar to the original study, all the participants were provided educational material about the research topic.

The developers were quite experienced, where 21% had worked for more than 10 years in software development, and only 21% had fewer than five years of experience. Fully 56% of the respondents had a bachelor's degree, and only two developers had no formal higher education.

We cannot share the rest of the characteristics of the developers and the company due to confidentially reasons (such as gender, software system type, and programming language).

The third phase (in comparison with section 4.1.3) consisted of two steps of data collection using two sets of surveys. The first survey gathered similar descriptive statistics to summarize the backgrounds of the participants as were collected in the original study. The second step collected data using a similar longitudinal research design (one survey per week). This stage was designed to collect reported data from 47 software developers.

However, the replication survey was designed slightly differently, compared to the survey that was used in the original study. Each week, the respondents were asked to report on *one specific major work item* within their ongoing project and both how much time the respondents spent on this item and how much time was wasted due to experiencing TD for this specific item. More specifically, the participants were asked to report the share of their total development time spent on the specific work item and the share of that time they wasted due to TD as well as what extra activities the wasted time was spent on and the source of the problem for which they wasted time on this work item.

The fourth phase (in comparison with section 4.1.4), the analysis of the collected data in the replicated study has been analyzed in quantitatively, that is, by interpreting the numbers collected from the survey answers. The replicated phase did not include steps similar to the fifth and sixth phases we conducted in the original study, where we conducted supplementary qualitative semi-structured interviews and analysis of those.

In total, we received data from 177 different working items, and the results of the replicated study and its comparison with the original study are presented in Section 5.7.

5 Results and Findings

The following subsections present the results for the research questions presented in Section 2,

and the results are grouped according to each research question.

5.1 WASTED TIME

The first set of questions (RQ1.1, RQ1.2, and RQ1.3) focused on how much of software developers' overall development time is wasted due to TD and whether the distribution of the wasted time varies with developer characteristics and follows any patterns.

5.1.1 Wasted time (RQ1.1). During the longitudinal data collection phase, 43 developers reported their wasted working time due to experiencing TD twice a week for seven weeks (in total 473 data points). On average, each developer reported 10.7 times out of 14 possible occasions (with a median of 12 and standard deviation of 3.9 times) with the average time interval between the reporting occasions of 3.1 days, with a median of 2.7 and standard deviation of 1.3 days (excluding Saturdays and Sundays).

The single most striking observation to emerge from the data was that the respondents reported that, on average, **23.1%** of all software development time is wasted due to TD, with the standard deviation of 21.1% and a median value of 17.13%.

When calculating the average amount of wasted time, the different interval lengths between the occasions were taken into account. This meant that, for example, when a developer reported 33% waste for a three-day period following a reported waste of 40% for a 10-day period, the average wasted time was calculated as 38.38%.

Turning to the distribution of the reported wasted time, Fig. 4 shows a histogram of the respondents and their reported wasted time from the longitudinal data collection phase. Most respondents wasted between 0% and 10% of their working time. It is interesting that six respondents reported a wastage of more than 50% of their working time.

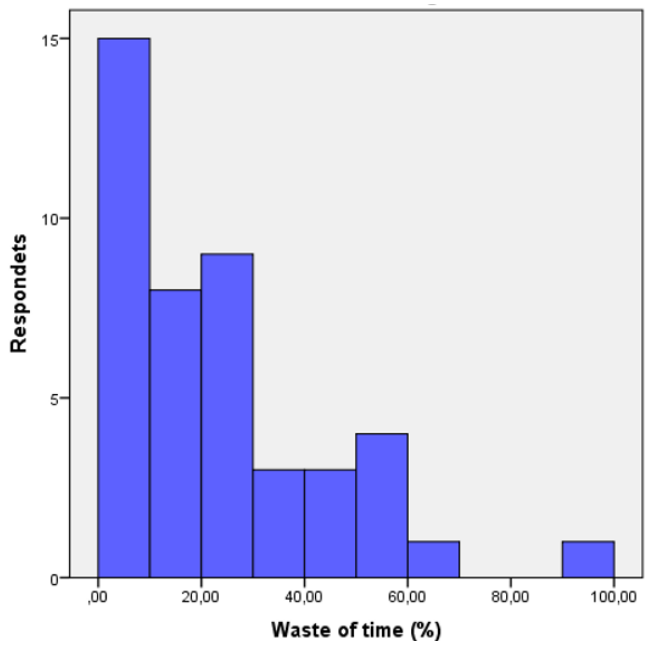


Fig. 4. Distribution of the reported wasted time

To put the above numbers into context, Fig. 5 shows an overview of the distribution of the wasted time as a function of calendar time as reported in the longitudinal data collection phase.

The mean wasted time remained quite stable over the entire study period at about 25%.

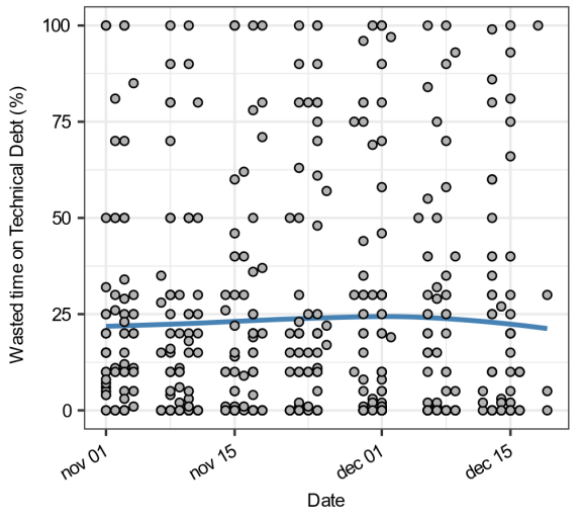


Fig. 5. Wasted time on technical debt as a function of calendar time. The blue curve, presenting mean waste as a smooth function of time, was estimated using LOESS.

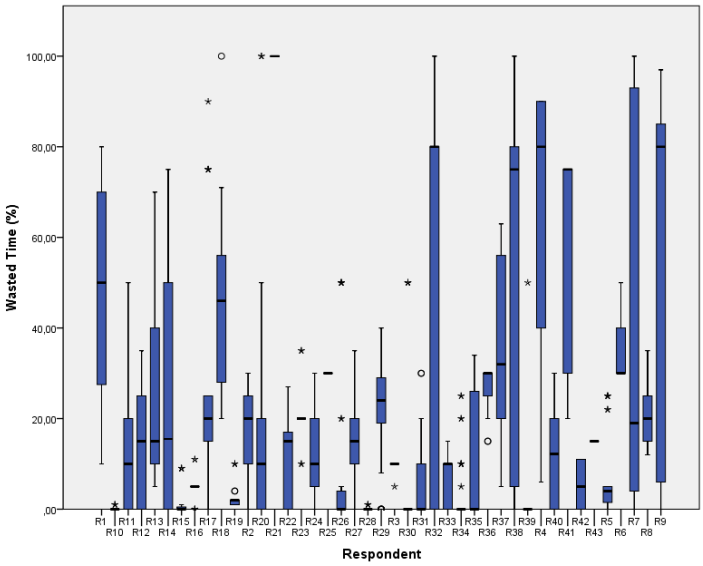


Fig. 6. Distribution of the wasted time for each respondent

Fig. 6 shows the distribution of each respondent's reported wasted time from the longitudinal data collection phase, illustrated by a boxplot. Looking at the comparison of medians (the bold horizontal black line in each box) among the different respondents in the figure, it is apparent that the different respondents waste different amounts of time due to experiencing TD. The figure also demonstrates that there are different distances between the median values and the upper and lower quartile among the different respondents, which indicates that, among the respondents, there are variances in terms of how consistently the wasted time is reported. Some respondents' amounts of reported wasted time vary greatly over time, while other respondents' amounts of reported wasted time are more consistent and concentrated.

5.1.2 Characteristics (RQ1.2). When examining the distribution of the wasted time with respect to different characteristics, we focused on the different variables: (a) years of experience as a developer, (b) gender, (c) level of education, (d) programming language, (e) company, (f)

age of the software, and, (g) type of software.

The used variables were the same variables that were collected and assessed in the first step in the data collection phase (e.g., the start-up survey).

The percentage of wasted time versus related to various subject characteristics, from the longitudinal data collection phase, is presented in Fig. 7 and Fig. 8. As illustrated in Fig. 7, three variables showed substantial differences with respect to the reported amount of wasted time:

- The company, explaining 20.4% of the variance in wasted time, with an average waste ranging from 18.1% in company A to 51.5% in company G (Fig. 7a).
- Software age, explaining 17.4% of the variance in wasted time, with an average waste ranging from 15.3% (software 5 - 10 years old) to 55.3% (software > 20 years old, n = 2) (Fig. 7b).
- Type of software, explaining 33.3% of the variance in waste time. The wasted time was below average for Modelling and Simulation Systems, Real-Time Systems and Embedded Systems, and more than twice the average for Data Management Systems, System Integration Web 2.0 / SaaS Systems (Fig. 7c).

As illustrated in Fig. 8, there were only small variations in the reported amount of waste time with respect to experience, gender, level of education and programming language.

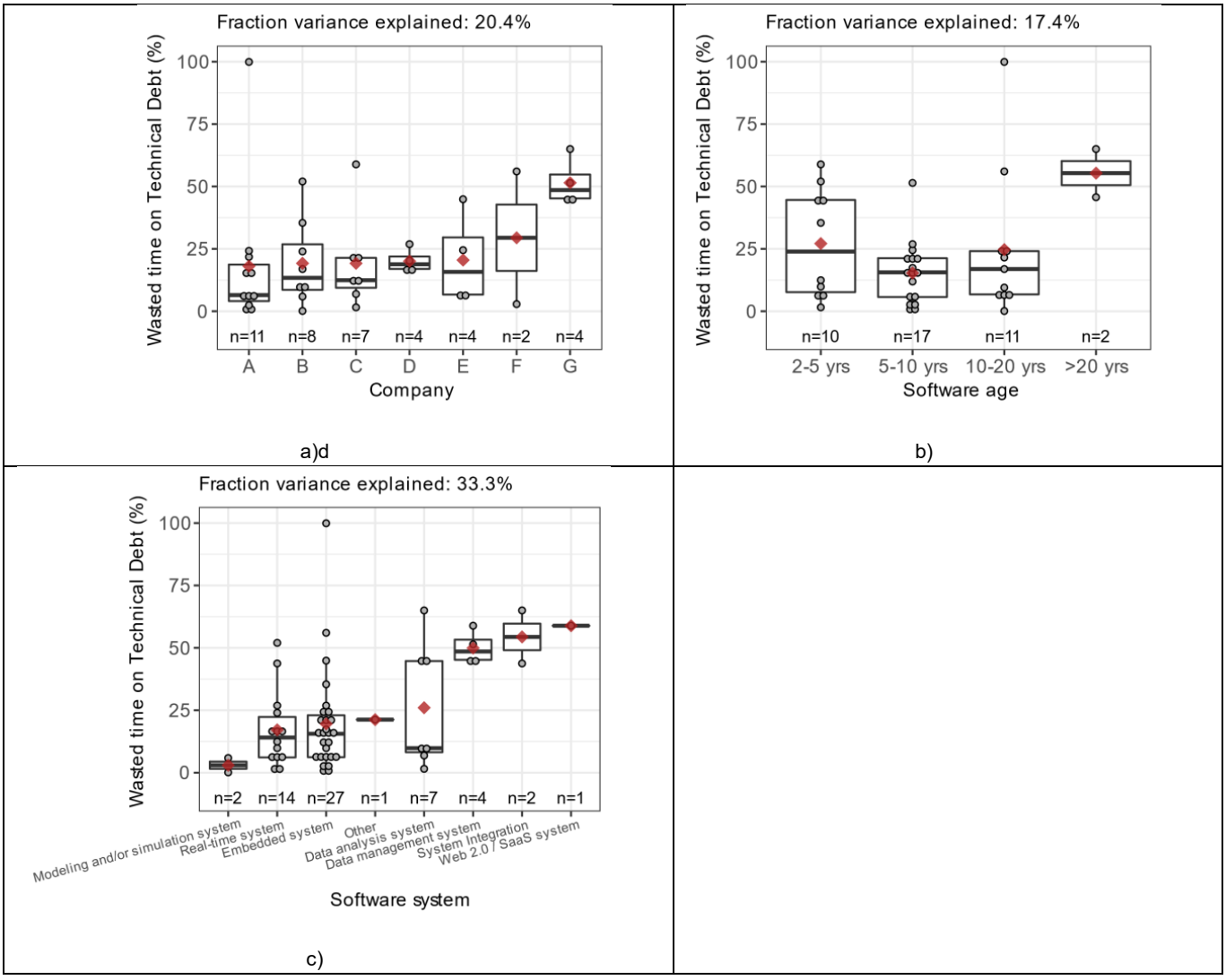


Fig. 7 (a, b, c): Percentage wasted due to technical debt vs. (a) company, (b) software age, and (c) software system. Circles

represent individual data points, binned into 50 distinct intervals along the y-axis. The red diamonds show the mean within each group. For a software system, individuals may appear in multiple groups. Means and fraction variance explained are computed by ordinary least squares regression, taking concurrent software systems into account.

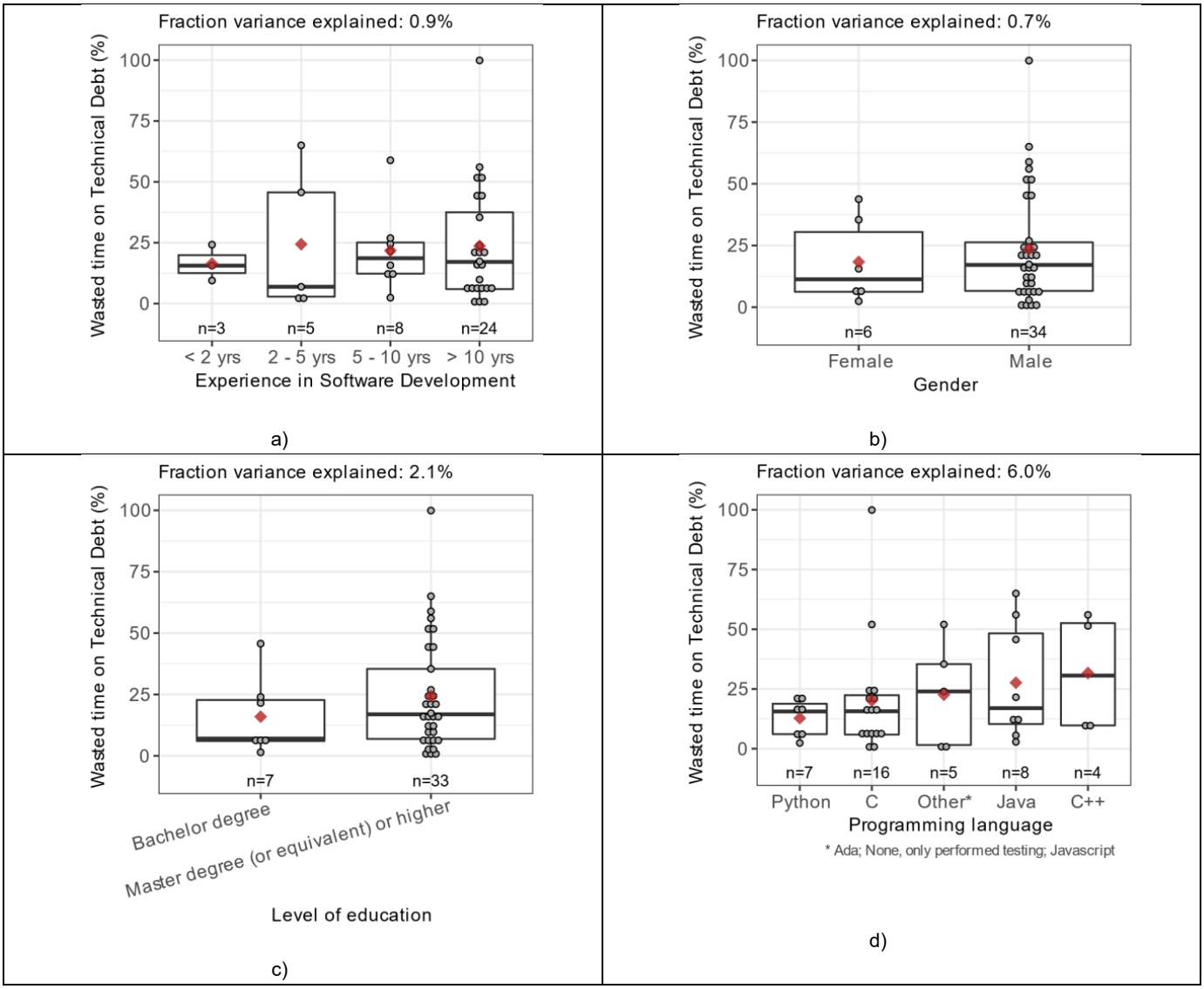


Fig. 8 (a, b, c, d): Percent wasted time on technical debt vs. experience in software development (a), gender (b), level of education (c), and programming language (d). Circles represent individual data points, binned into 50 distinct intervals along the y-axis. The red diamonds show the mean within each group. For the programming language, individuals may appear in multiple groups. Means and fraction variance explained are computed by ordinary least squares regression, taking concurrent programming languages into account.

Developer characteristics were further evaluated in multivariable analyses using non-parametric regression trees, aiming to find combinations of characteristics associated with greater or lower waste. However, only trivial models consisting of a single variable were found, as adding more variables resulted in increased cross-validation error. This is probably due to the limited sample size and more specifically to the small number of individuals in subgroups with high waste.

5.1.3 Distribution (RQ1.3). When examining each respondent's distribution of the wasted time over the study period, we noticed that the variations in mean level and trend during the

study period differed between the respondents. All respondents showed an individual 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 from the longitudinal data collection phase, are illustrated in Fig. 9. For some developers, it was evident that the wasted time varied *periodically* over time, meaning that, within a sub-period, the distribution followed a low, high, or fluctuating pattern, but, at some point, 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. The *Low* and *High* profiles illustrate a wasted time that is largely consistent over time, even if some peaks can be recognized.

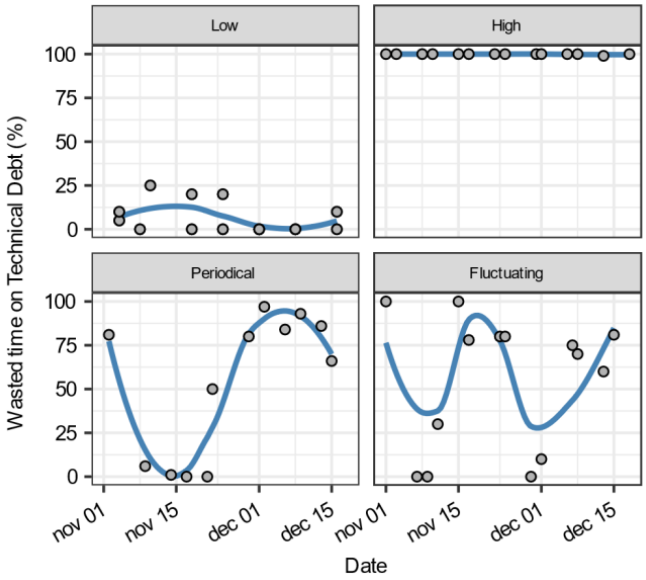


Fig. 9: Different distribution profiles of the wasted time. The blue curves, presenting mean waste as a smooth function of time, were estimated using LOESS.

The most common pattern among the respondents was the *Low* profile, and the less common pattern was the *High* profile. The distribution between the *Fluctuating* and the *Periodical* patterns was largely equally divided.

Finding 1: *Almost a quarter of all developers' working time is reported as wasted due to having TD.*
Finding 2: *Company and System types have the strongest impact on the amount of wasted time.*
Finding 3: *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.*

5.2 Additional Activities

The next research question (RQ2) explores the different activities on which the wasted time is spent and also whether the amount of the wasted time relates to any specific activity.

When developers encounter TD during their software development work, they are forced to perform supplementary actions. Accordingly, these different activities would not have been necessary if the TD were not present.
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. For each of the 473 reporting occurrences, the respondents selected the activities on which the wasted time was spent from a list of pre-defined options (listed in Section 4.1.2).

The distribution of wasted time across different activities, from the longitudinal data collection phase, is presented in Fig. 10, with activities sorted according to mean waste per activity. As illustrated in this Figure, the mean wasted time was greatest when performing *Additional Testing*, with a mean wasted time of 43.1%, followed by *Additional Refactoring* (mean waste 42.8%) and *Additional Code Analysis* (mean waste 37.9%). The activity with the weakest association to the wasted time is *Additional Communication*, with a mean waste of 28.8%. The "*None*" option was mainly chosen when no time at all was wasted (mean waste 5.4%).
The marginal effect of each activity, accounting for concurrent activities and subject effects through mixed effects models, is presented in Table 2. The activity with the strongest effect on wastage of time was again performing *Additional Testing,* associated with a waste increase of 12.4 percentage units (p.u.) (95% CI 8.7 to 17.3 p.u.), followed by *Additional Code Analysis* (mean waste increase 11.7 p.u) and *Additional Refactoring* (mean waste increase 10.3 p.u.). Again, performing *Additional Communication* was associated with little additional waste (mean waste increase 4.1 p.u., 95% CI 0.9 to 8.0 p.u.). This means that having to perform *Additional Code Analysis* increases the average amount of wasted time by 12.4%, compared to if no additional code analysis had to be performed. When selecting among the different activities the wasted time was spent upon, the respondents also could enter an additional activity manually in a text field that was not predefined, and, interestingly, no other additional activities caused by the present TD were added here by the respondents. This implies that the six listed activities cover most of the extra activities on which the time is wasted due to experiencing TD.

One interviewee described Test TD as the need for performing additional testing "*if you have made a change, you have to spend a little more time verifying it manually, as there may not be automatic tests to verify it and to ensure that everything still works after the change. And yes, it may also be that everything may not be developed personally by the developer and therefore may require some extra tests to be run before making the change."*

The additional code analysis was described by one interviewee as *"first you have to understand the code and then take input and discuss the code with colleagues and only after that, one can actually do something about the code"*

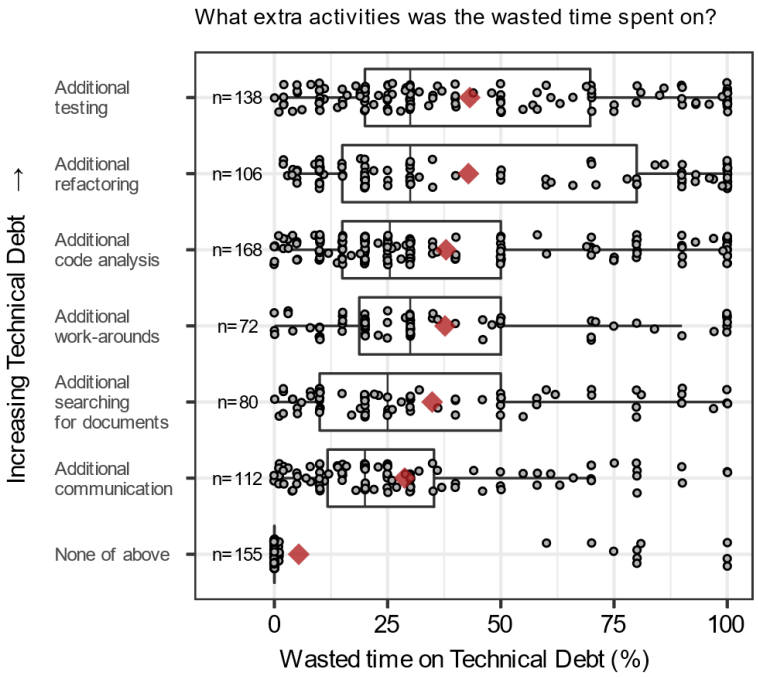


Fig. 10: Wasted time due to technical debt vs. Activities. Circles represent individual data points, binned into 50 distinct intervals along the y-axis. The red diamonds represent the mean waste for each activity.

TABLE 2
THE AVERAGE EFFECT OF TD ACTIVITIES ON WASTED TIME

What extra activities was the wasted time spent on?	Estimated effect on waste increase* (95% CI)**
Additional testing	12.4 (8.7; 17.3)
Additional code analysis	11.7 (7.8; 16.0)
Additional refactoring	10.3 (6.4; 15.2)
Additional searching for documents	6.5 (2.3; 11.6)
Additional work-arounds	6.3 (1.7; 10.1)
Additional communication	4.1 (0.9; 8.0)

* Estimated effects are presented in percentage units.
** 95% confidence intervals were computed using the non-parametric bootstrap percentile method, using 10000 bootstrap replicates.

Finding 4: *The activity "Additional code analysis" has a consistent association with wasted time (across both studies, see later section 5.7.2) with "Additional testing" showing up strongly in the primary study.*

5.3 TECHNICAL DEBT TYPES

Since there are several types of TD [52], and these different TD types could have different levels of negative impact on the amount of the wasted time, and these different TD types could potentially have different levels of negative impact on the amount of the wasted time, in this third research question (RQ3), we sought to explore the ways in which these TD types impact 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."

From the data in Table 3, it is apparent that a significant proportion of the TD encountered is related to source code, where 19.3% of the code-related TD is encountered "To a great extent."

TABLE 3

THE 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%

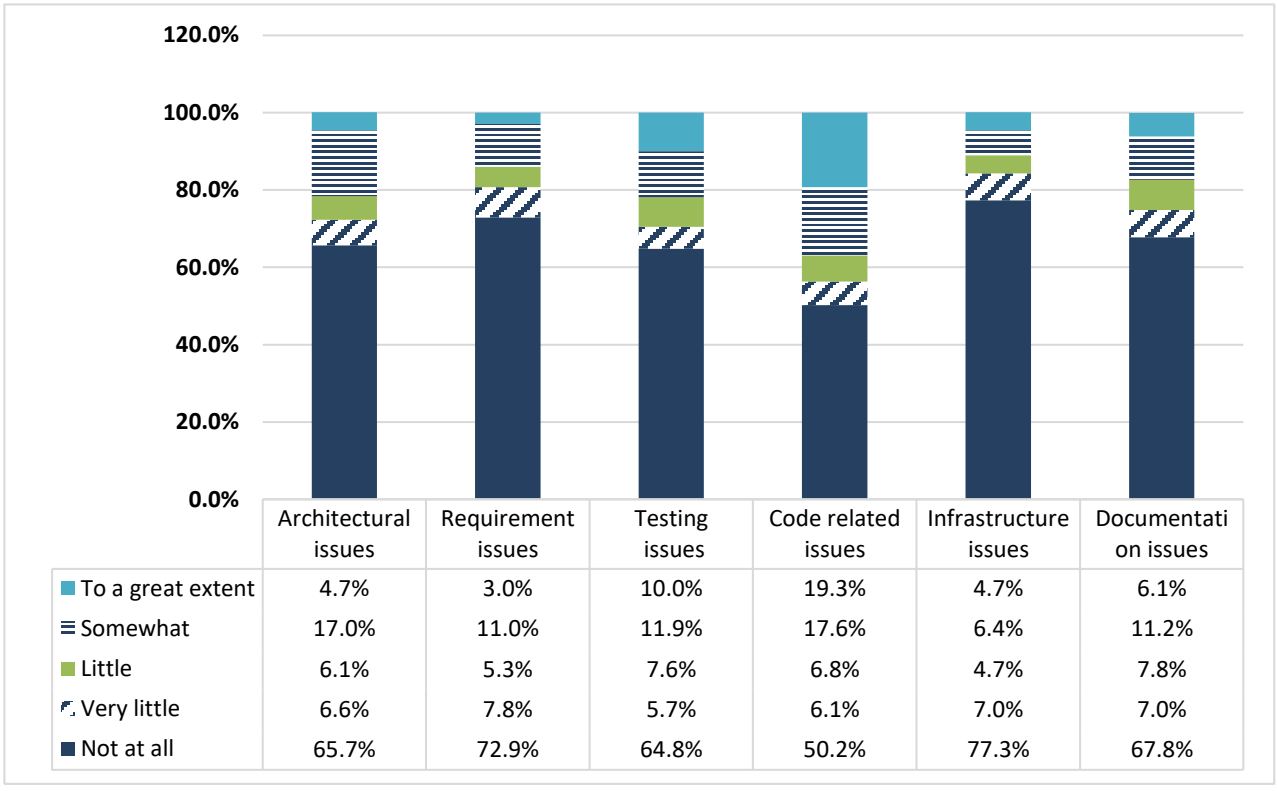


Fig. 11 illustrates each studied TD type and its relation to the reported amount of wasted time from the longitudinal data collection phase. A positive trend was observed between increased levels of encountering the different TD-types and increased average waste—meaning that the more of each TD type the developers encounter, the more time they waste—although a monotonic trend was not observed for all TD types. The strongest association with wasted time was observed for *Testing* issues, explaining 21.2% of the variance in wasted time, followed by *Code-related issues* and *Architectural issues.* Only a weak association was observed between wasted time and *Requirement issues*, explaining only 9.0% of the variation in waste. We, therefore, suggest that the association of Requirement TD and the amount of wasted time be investigated further in future studies.

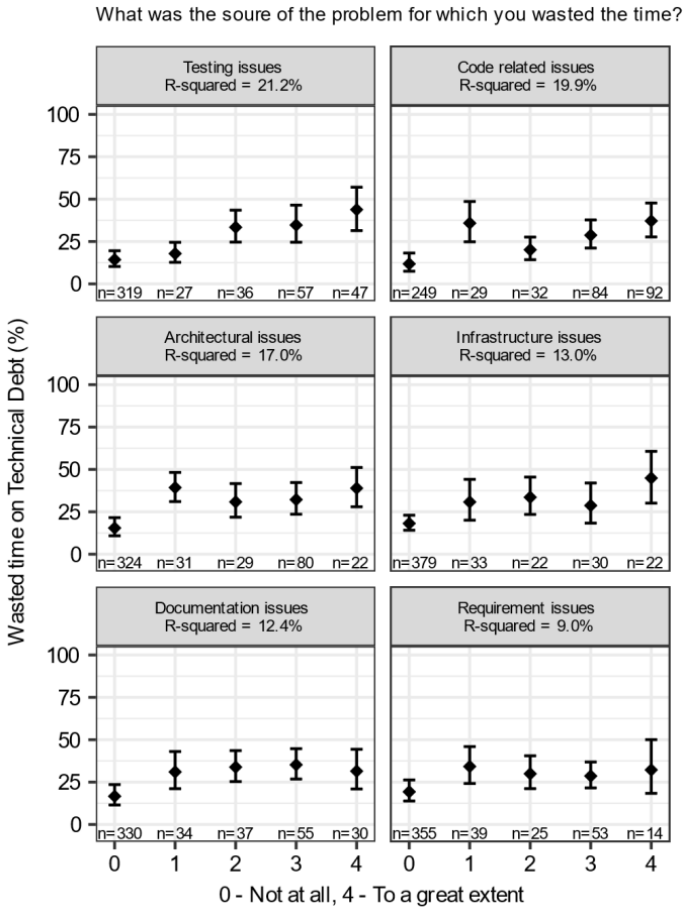


Fig. 11: Wasted time on technical debt vs. technical debt type. The black diamonds represent the mean for each level on the Likert scale, error bars present 95% confidence intervals for the mean.

Finding 5: *The TD type "Testing issues" has the strongest association with the wasted time, followed by "Code-related issues" and "Architectural issues," with increased level of negative impact associated with increased amount of wasted time. Only a weak association between Requirement TD and amount of wasted time was observed.*

5.4 Introducing new Technical Debt

Sometimes, developers are forced to introduce new additional TD due to already existing TD. The interest payment could take place in the form of, for example, introducing new shortcuts and maintenance obligations taken as the developer tries to fix the prior debt.

This research question (RQ4) aims to address the amount of additional TD developers are forced to introduce due to present TD.

The result in Fig. 12 graphically illustrates that in 24% of all the reported occasions the developers reported during the longitudinal data collection phase, that they were, to some extent, forced to introduce additional TD. Within these 24% reported occasions, on 13% of the occasions the respondents reported that they were forced to introduce additional TD "To a very little extent," and 3.6% reported "To a little extent," and 4% "To some extent," and 3% reported that they were forced to introduce additional TD "To a great extent."

When performing a detailed analysis of each reported occasion where the respondents were forced to introduce new additional TD due to already existing TD "To a great extent," the result shows that the encountered TD types for these occasions were Test TD (in 53% of the occasions) and Source Code TD (in 47% of the occasions).

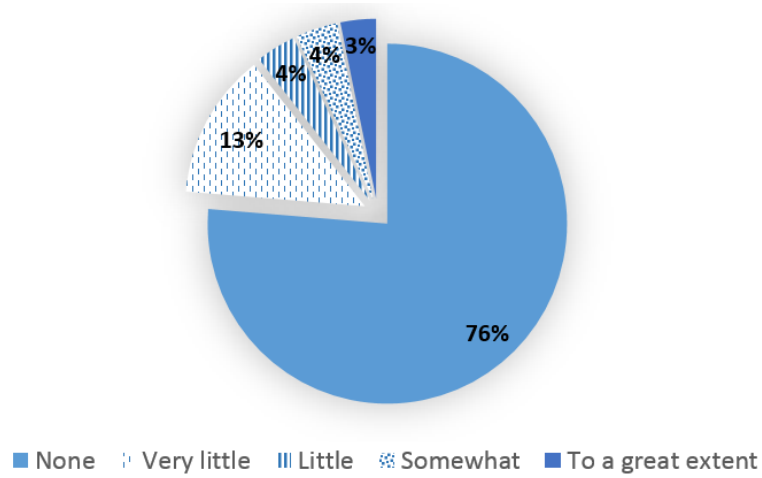


Fig.12. Introduction of new TD

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."* Moreover, in discussing this issue, another interviewee said, "*You implement suboptimal solutions because you are in a hurry. Plus, we don't find the time to*

Finding 6: *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.*
Finding 7: *Encountering Test TD and Source Code TD forces the developers to introduce additional TD to the largest extent.*

use the test tools we have and write tests for everything."

5.5 Awareness and Benefits

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

Apart 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 after that, we showed them their results from their 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 similarly. The benefits were described by the developers as the quantified wasted time potentially helping to detect and to predict the need for additional quality improvements.

Some developers also highlighted the benefits of being able to improve forecasting and capacity planning and being able to justify change and thereby motivate 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.*"

One developer in the study also described the benefits of tracking the wasted time as a useful tool when implementing a new type of management strategy with the goal of decreasing the negative impact of TD. "*You could combine it [the reporting of the wasted time] with working in another way. Because you can use it for tracking…We should also change the way that we actually make new things to try to work without creating new technical debt. And then you use the tracking so if these working methods actually work*." 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 a fact.*"

Finding 8: *Developers have a higher awareness than their managers of how much time is wasted due to TD, and the developers and their managers seem to appraise the amount of development time that is reasonable to waste due to TD in different ways.*
Finding 9: *Both developers and managers described the benefits of knowing the amount of wasted time in a similar manner. The amount of wasted time was found to be a useful indicator of the software quality, and it could also be used for improving forecasting and better capacity planning and assisting the communication between managers and their developers.*

5.6 Challenges of tracking TD interest

This question (RQ6) highlights the interaction between developers and managers with regard to the challenges of tracking the interest of TD.
Overall, even though developers consider themselves to benefit from making the amount of wasted time evident, most of the developers did not have a positive attitude toward continuously reporting the wasted time to their managers. Even if the interviewed developers generally claimed that the specific reporting task took them only a couple of minutes for each reporting occasion, several interviewees argued that this reporting task would require unwanted additional time and effort.
Similarly, even if the managers consider the benefits of knowing the amount of wasted time to be important, the managers did not explicitly request this information from the developers, and, in general, they did not have a positive attitude to asking the developers to report their wasted time.
Several managers expressed their unwillingness to introduce new reporting tasks to the developers, and one manager described the fear of causing extra stress for the developers. *"There is a certain fear of reporting. It will almost become a negative spiral of it eventually. We have little stress-related sick leave. If you get that, because of such a system, you probably have not achieved that much."* Another manager echoed this notion, describing the attitude toward introducing reporting of wasted time to the developers. *"Even though I see the value of this kind of data, it's nothing I'm going to force anyone to report, but if people are interested in continuing this here, I'm very welcome from my side, but it may be on an interest-based basis."*
Even if all interviewees were familiar with the concept of TD and its related negative effects, this knowledge was not put into practice, and the lack of having an overall strategy for managing TD was evident.

Finding 10: *The willingness to quantify the wasted time is a major challenge, since developers and managers do not, in general, have a positive attitude toward implementing additional reporting.*
Finding 11: *None of the interviewed companies had a clear strategy on how to track and address the wasted time.*

5.7 Result of the replication study

The motivation to carry out this replication part of the study is to investigate further whether the previous results demonstrate that the findings can be repeatedly generated and thus the original findings were not an exceptional case. In particular, the aim is to broaden the results obtained in RQ1.1, RQ2, and RQ3 by investigating the same research areas but with other independent data sets, as described in section 4.1.7.
Each sub-section will first report the results from the replicated study and then present a comparison of its results with the results of the original study.

5.7.1 Replication of result addressing wasted time due to technical debt

Research question RQ1.1 addresses how much of software developers' overall development time is wasted due to technical debt. The replicated study focuses on how much time was wasted on specific working items.
In total, we have analyzed data from 177 reported working items from 47 developers at the same company but working on different projects and with different products.

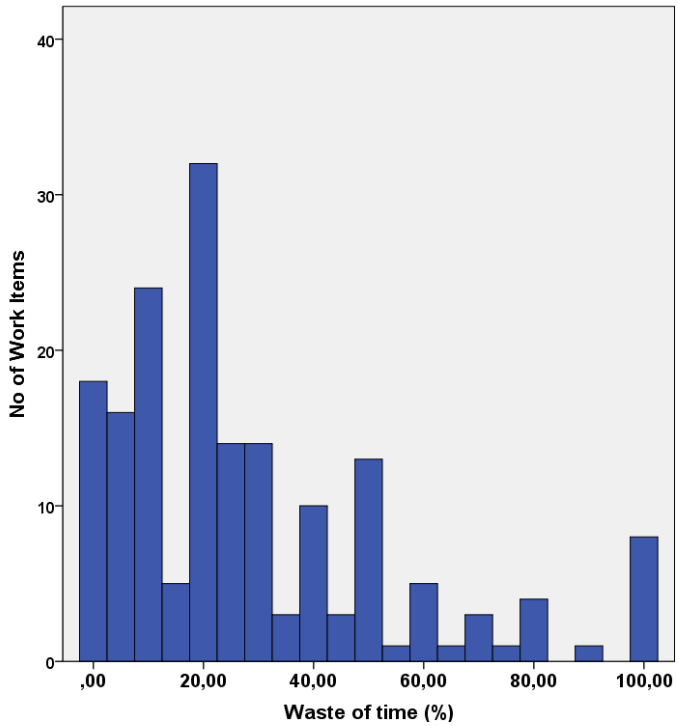


Fig 13. Distribution of the reported wasted time from the replicated phase

The replicated study found that, on average, 28.51% of the software development time for the reported working tasks is wasted due to TD, with the standard deviation of 25.37% and a median value of 20.0%. Fig. 13 shows a histogram of the reported wasted time for each work item. Most work items wasted, on average, 20% of their time due to experiencing technical debt.

Even if both data sets from the original and the replicated study are large enough to support comparison, they do not permit comparison using the same statistical methods since they are based on different variables. However, the result of the two studies can be examined together

by studying the result of the reported amount of wasted time.
In the replicated study, the respondents were asked to report on the wasted time for the work item they spent most of their working time on since the last time they took the survey. However, since the respondents potentially could perform other working tasks during the period (for which we do not know the amount of wasted time), we cannot generalize their reported waste of time for the full period. Our analysis of the time the respondents spent on the reported work items shows that the respondents spend on average 57.63% of their working time on each of reported work item.
Despite the different used variables, one might intuitively expect that the amount of wasted time due to TD should be quite similar in both of the studies. In the original study, the respondents reported that, on average, 23.1% of all software development time is wasted due to TD. This result is a slighter lower share of wasted time than the reported average of the replicated study, where the respondents reported that, on average, they waste 28.5% on each of the reported tasks. However, by combining results from the original study with results from the replication study, we conclude that Finding 1 stating that "*Almost a quarter of all developers' working time is reported as wasted due to having TD*" is valid and strengthened.

5.7.2 Replication of result addressing different activities

This part of the replication study aims at replicating the findings from RQ2 by exploring the different activities on which the wasted time is spent and also whether the amount of the wasted time relates to any specific activity.
The distribution of wasted time across different activities from the replication study is presented in Fig. 14 where the activities are sorted according to mean waste per activity. By studying the ranking of activities with respect to average waste of time in this figure, it is evident that this data differs noticeably from the data in the original study. For instance, in the replicated study, the activity of performing "*Additional code analysis*" has the strongest association to the wasted time whereas, in the original study, performing "*Additional Testing"* has the strongest association. In the replication study, the additional activity of performing "*Additional Testing"* is ranked quite differently since it has the weakest association with the wasted time (except for the "other activity option").
However, one should note that the mean value of each of the activities is fairly close to the original study, except for the mean value of "*Additional Testing."*

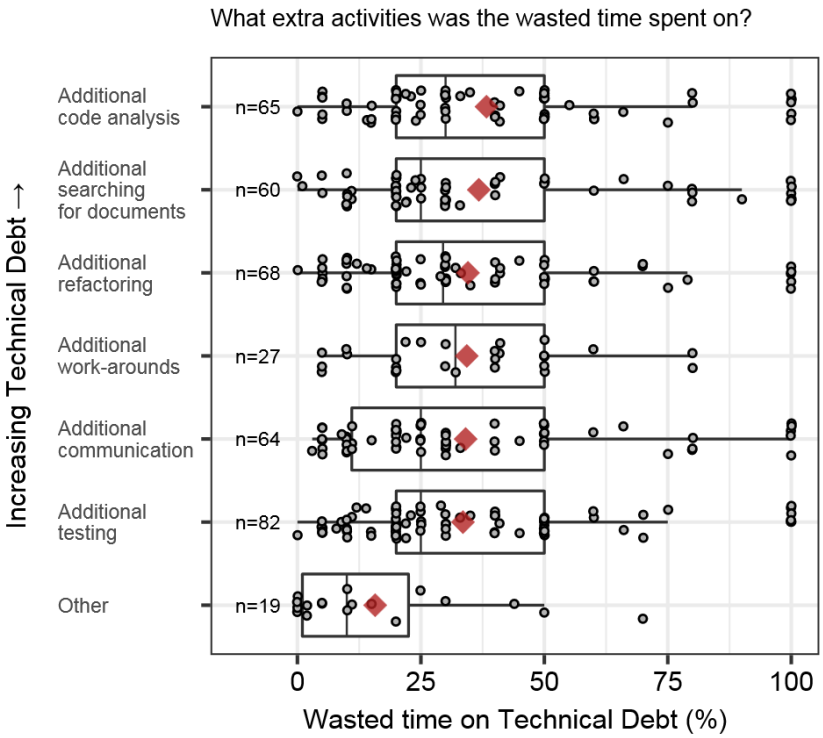


Fig. 14: Wasted time due to technical debt vs. activities for the replicated study. Circles represent individual data points, binned into 50 distinct intervals along the y-axis. The red diamonds represent the mean waste for each activity.

In a similar way as in the original study, the marginal effect of each activity, accounting for concurrent activities and subject effects through mixed effects models, is presented in Table 4. The activity with strongest effect on wastage of time in the replicated study was again performing *Additional code analysis,* associated with a waste increase of 10.4 percentage units (p.u.) (95% CI 3.9 to 18.2 p.u.), followed by *Additional searching for documents* (mean waste increase 8.7 p.u) and *Additional Refactoring* (mean waste increase 8.6 p.u.).

TABLE 4
THE AVERAGE EFFECT OF TD ACTIVITIES ON WASTED TIME FROM THE REPLICATED STUDY

What extra activities was the wasted time spent on?	Estimated effect on waste increase* (95% CI)**
Additional code analysis	10.4 (3.9; 18.2)
Additional searching for documents	8.7 (-0.0; 17.1)
Additional refactoring	8.6 (1.6; 15.8)
Additional work-arounds	6.9 (-3.2; 15.0)
Additional communication	1.8 (-4.1; 9.2)
Additional testing	1.8 (-3.8; 8.9)
* Estimated effects are presented in percentage units. ** 95% confidence intervals were computed using the non-parametric bootstrap percentile method, using 10000 bootstrap replicates.	

To conclude, the original and the replicated studies show a somewhat different relationship and ranking between the amounts of wasted time in relation to the different activities, even if the

mean value of each individual activity is quite similar to the values in the original study.

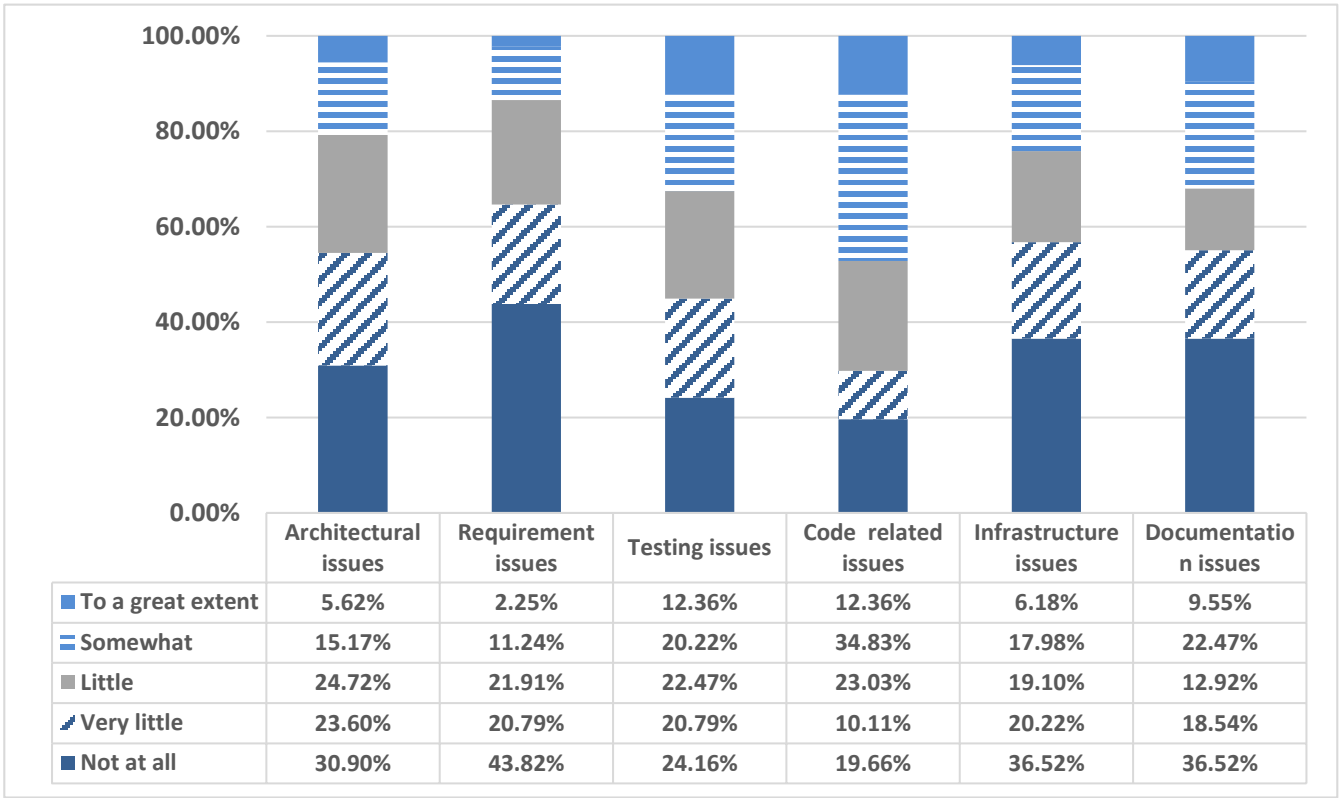
5.7.3 Replication of result addressing technical debt types

When answering research question RQ3, we examined the extent to which different TDs were encountered in order to understand their frequency.

TABLE 5

THE LIKERT SCALE OF EACH ENCOUNTERED TD TYPE

	Architectural issues	Requirement issues	Testing issues	Code related issues	Infrastructure issues	Documentation issues
Not at all	30,90%	43,82%	24,16%	19,66%	36,52%	36,52%
Very little	23,60%	20,79%	20,79%	10,11%	20,22%	18,54%
Little	24,72%	21,91%	22,47%	23,03%	19,10%	12,92%
Somewhat	15,17%	11,24%	20,22%	34,83%	17,98%	22,47%
To a great extent	5,62%	2,25%	12,36%	12,36%	6,18%	9,55%



As shown in Table 5, a significant proportion of the encountered TD is related to source code and testing, where 12.36% of the TD for those types is encountered ‘‘To a great extent.’’

Even though the two sets of results are not identical, they share the same finding when studying which TD types are most often and most seldom encountered to a large extent. When comparing the result from the original study with the result from the replicated study, it is apparent that the code-related TD and test related TD are the two TD types that are most often encountered “To a great extent.” Furthermore, the results in both studies show that requirement related TD is most seldom encountered “To a great extent.”
To conclude, these replicated results confirm previous findings and contribute additional evidence that developers suffer from several different TD types and that they differ in terms of their frequency and magnitude.

6 Discussion

The following subsections present discussions and limitations for the research results presented in Section 5, and the results are grouped according to each research question followed by a section addressing the replicated phase of the study and, finally, a section about the implications for practitioners and researchers.

6.1 Wasted Time and Introduction of new TD

The first three research questions (RQ1, RQ1.1, and RQ1.2) 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, where the ratio of development effort and maintenance effort in a product development lifecycle becomes tilted toward more maintenance effort due to the presence of TD in the software.
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. 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 circle of TD growth. This result quantitatively corroborates the findings of [26], where the term contagious debt is described.

6.2 Additional Activities

The second research question (RQ2) in this study sought to explore which activities the wasted

time was spent and also whether 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 code analysis” across both the primary and the replication study with “Additional testing” showing up strongly in the primary study.
This result implies that, if the systems did not have TD, the time spent on these activities could be reduced. Furthermore, spending a great deal of time on these activities during the software development could, consequently, potentially be an indicator of a system suffering from TD and also an indicator of the amount of interest that has to be paid and thereby indicate a decrease in developer productivity.

6.3 Technical Debt Types

The results from the third research question (RQ3) show that all TD types are significant and strongly associated with the amount of the wasted time, whereby *Source code TD* has the strongest association with the amount of wasted time. This result demonstrates that all the 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 its negative impact as more prominent. Likewise, the results show that developers encounter less Requirement TD and Infrastructure TD, which also points to the idea that developers are less prone to attribute 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 is currently the case, focus primarily on code-related TD.

6.4 Awareness and Challenges

The fifth and sixth research questions (RQ5 and RQ6) address the levels of awareness of the developers and their manager regarding the amount of time wasted due to TD, the benefits of this insight, and how they 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 the professions also seem to have different views on what is a reasonable or unreasonable amount of time to waste on TD. Both developers and managers could see the benefits of quantifying the amount of wasted time, but both professions were, in general, reluctant to practically implement a systematic approach to quantifying the wasted time. Developers argued that reporting the wasted time due to TD would require additional time and effort, while the managers were hesitant to implement such measures due to the extra workload it would place on the developers. 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 lack of developer productivity could end up being increased instead of being reduced. In the long run, low developer productivity can, due to time

pressure, stress the developers to further introduce new TD and can also have a negative impact on the amount of new features that can be implemented and can harm both the maintainability and evolvability of the software product.

6.5 Replication study

In general, a successful replication of a study is one that helps the research community gain information about conditions under which the results hold [53], [54].
The replication phase of this study aimed at replicating the results addressing RQ1.1, RQ2, and RQ3. The replicated results for RQ1.1 and RQ3 where, overall, in line with the results from the original study, which implies greater reliability and confidence in these results.
Further, the results for RQ2 contradicted to some extent the results from the original study addressing the different activities on which the wasted working time was spent.
A hypothetical explanation of this discrepancy may be found in how the data was collected in the replicated study. One of the major alterations of the data collection in the replicated study is that it was collected at only one company (compared with six companies in the original study). This replication setting creates a context in which the sources of variations potentially could be limited. One could expect that several of the participating developers in the replication study worked in a similar environment or even in the same code base, thus experiencing TD that required specific activities to take place, resulting in a company-specific result that is less generalizable to a broader community.
However, even if the results from this research questions show a different ranking of how strong each activity is associated with the amount of wasted time, the mean value of each activity was quite similar to the original results. In the replication study, the result showed that performing additional code analysis had the strongest association with the amount of wasted working time due to experiencing TD, as compared with the original study where performing “Additional Testing” had the strongest association. This result could potentially be explained by a company-specific environmental setting, but further studies that take these variables into account will need to be undertaken.

6.6 Implications for Practitioners and Researchers

It is commonly quite difficult to motivate and argue for the need to prioritize refactoring activities due to software experiencing TD in today’s software industry. One major reason for this can be described in terms of the lack of knowledge about how TD negatively affects the software developer productivity.
This study has shown that an extensive amount of valuable working time is wasted due to TD and that this TD causes the developer to perform different activities that would not have been necessary if the TD were not present.
However, being able to describe and understand the amount of the negative effects of TD in terms of wasted time can help when developers argue for the need to initiate refactoring to reduce the amount of TD and thereby potentially decrease the future amount of time wastage.
This study makes a novel contribution to the existing body of knowledge and suggests several

important practical implications that demonstrate the impact TD has on software development productivity. This contribution of this study can be used by both software practitioners and researchers within the field:

- Based on this study's empirical result, we show that software developers report that they waste, on average, 23% of their working time due to TD.
- 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.
- 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.
- 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 or unreasonable amount of time to waste due to TD.
- 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.
- This study shows that both developers and managers see the benefits of tracking the amount of wasted time, but both professions are somewhat reluctant to implement such measures in practice. This unwillingness is recognized as a challenge for the companies.
- 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.
- Overall, these findings suggest strong recommendations for software companies to focus further on continuously undertaking refactoring initiatives of TD issues to keep the amount of TD at bay on an ongoing basis. In general terms, this means that such TD remediation and prevention initiatives also would have a positive impact on the overall developer productivity.

7 Verifiability, Limitations, and Threats to Validity

The purpose of this section is to reflect on the extent to which this study has addressed the goal of ensuring verifiability, describing limitations, and finally addressing potential threats to validity.

7.1 Verifiability and Limitations

There are several important limitations that necessitate a cautious interpretation of the results of the present study. First, selection bias is a potential limitation since the data from the invited companies in the study was gathered only in specifically chosen companies. Second, given the self-reported nature of the collected data in the surveys, 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. Further, the participants did not use any supporting software tools to identify the reported TD and the participants' accuracy of identifying TD were neither evaluated.

However, one could argue, since reporting the time spent on different tasks and activities is a common practice for developers performing their time registration, their ability to report the time should be reasonably sound. With this as a background, together with the provided education material, guiding the participating developer to distinguish between different types of TD would assist in determining the amount of wasted time and the specific type of TD on which it is being wasted.

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.

7.2 THREATS TO VALIDITY

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 explain accurately the phenomena that we observed [45]. To mitigate this threat, we have adopted both a univariable and a multivariable analysis of the data.

In this work, we have analyzed data to find a correlation between specific parameters in the reported data. However, we do acknowledge that this correlation does not imply causality between the variables and that the same results may not be reached if considering another collection of companies or developers with different characteristics. To further mitigate this threat, we conducted follow-up interviews with 12 of the participating developers in which the relationship between the reported amount of wasted time and the listed additional activities and the different TD types were assessed. However, the study design setting where the participants, first participate in the longitudinal phase of the study and thereafter are interviewed, can potentially introduce some bias into the data, and needs to be acknowledged.

Further, several of the findings were also validated by an additional replication study using a different and independent data sets, concluding and strengthening several of the derived results.

Furthermore, 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. In addition, in management research, it is not uncommon to use self-assessment when studying participants' productivity since this method is considered as a consistent method for objective measurements of performance [55].

The external aspect of validity addresses the extent to which it is possible to generalize the findings [35]. The responses that our respondents gave might not be representative of the entire

developer population. Although we cannot generalize the results, we can rely on a relatively high number of participating organizations (6), working in different business and application domains. Furthermore, in surveys, there is always a risk that the sample is biased, and, therefore, a potential threat relates to, for instance, the geographical, cultural, and demographic distribution of response samples. However, to confirm the generalizability of this study and to mitigate this threat, we have replicated the study with a different set of respondents, both in terms of geographical area and software development culture.

Construct validity addresses the extent to which the operational measures that are studied accurately represent what the researchers are considering [35]. This threat is related to whether we can correctly use the amount of wasted time as a substitute for software development productivity and whether the data collection approach is well-designed for the research purpose.

Using only a single report for each respondent involves a risk that this reporting gives a measurement bias [38]. To mitigate this threat, the data were collected using several reporting occasions over time, using a longitudinal data collection approach. This approach reduces the subjectivity of only studying the reported data on one single occasion.

Furthermore, this threat also relates to whether the study constructs are defined and interpreted correctly by the respondents [35]. 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 the educational material before starting the study. Another threat concerning the survey questions relates to whether the question could be clearly understood by the participants. To mitigate this threat, the initial survey draft was reviewed by all the authors, and we additionally made a pilot study with one software practitioner to examine the understanding of the survey questions.

The goal of reliability is to minimize the errors and biases in a study [45]. Reliability addresses whether the study would yield the same results if other researchers replicated them, following the same procedure, by means of the extent to which the analysis is dependent on specific researchers [35], [45].

To mitigate this threat in the original study, following guidelines by Yin [45], we designed the study in six distinct and separately documented phases (see Section 4.1), and made these steps as operational as possible to assist the repeatability of the study results.

As mentioned briefly in the above section, this study also includes an additional phase with the goal of replicating several of the findings. This replication study also assisted in mitigating several validity issues of the original study. In terms of external validity, the replication of the study assisted us by showing that several of the original results were not dependent on the specific conditions of the original study. The independence of the replicators from the original study lends additional confidence that the original results were not the result of data collection bias.

Similarly, in terms of internal validity, the replication study also assisted in showing the range of conditions under which the results hold. Since the several variables of the replication study were different from those of the original study (e.g., a new set of respondents, different data collection design, etc.), the replication phase contributed some confidence that the findings are not limited to the particular setting we had in the original study. Consequently, the additional

replication study addressed both external as well as internal validity.

8 Conclusion and Future Work

This study set out to analyze the negative effect TD has on software productivity from the point of view of software developers and their managers.

This study reports on the replication and extension of a longitudinal study of technical debt, where 43 developers reported twice a week for seven weeks how much time they waste due to TD, on which additional activities this time was spent, and what type of TD caused the wasted time.

This study provides evidence that TD hinders software developers by causing a substantial amount of wasted time. This wasted time negatively affects the development productivity and viability of the software. Even if both developers and their managers clearly see the benefits of reporting the wasted time, it is a challenge to implement such a reporting task due to unwillingness and time restrictions. This study shows that TD also contributes to the need to perform 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, and our results suggest that TD should be prioritized for refactoring because it forces the developers to introduce further additional TD, which generates even more interest.

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.

It was not possible in the present study to study the relationship between the qualities of the developers' software in relation to the wastage of their working time. However, as part of future work, we plan to extend this study by applying a triangulation of the quantum of TD within the investigated software system by, for instance, using tools for source code statistics, test statistics, or code churn metrics. This extension and replication of the study would provide additional heterogeneity in the relationship between TD and the productivity loss over different values of TD.

Acknowledgment

Many thanks to the industrial partners who participated in both the original and the replication study and interviews. We would also like to thank Henrik Imberg for his valued support during the statistical analysis of the data.

REFERENCES

[1] E. Tom, A. Aurum, and R. Vidgen, “An exploration of technical debt,” Journal of Systems and Software, vol. 86, no. 6, pp. 1498-1516, 2013.

[2] T. Besker, A. Martini, and J. Bosch, "Time to Pay Up - Technical Debt from a Software Quality Perspective." p. pp. in print. .

[3] P. Kruchten, R. L. Nord, and I. Ozkaya, “Technical Debt: From Metaphor to Theory and Practice,” Software, IEEE, vol. 29, no. 6, pp. 18-21, 2012.

[4] R. Mo, J. Garcia, C. Yuanfang, and N. Medvidovic, "Mapping architectural decay instances to dependency models." pp. 39-46.

[5] Z. Li, P. Liang, and P. Avgeriou, "Architectural Debt Management in Value-Oriented Architecting," Economics-Driven Software Architecture, pp. 183-204, 2014.

[6] C. Fernández-Sánchez, J. Díaz, J. Pérez, and J. Garbajosa, "Guiding flexibility investment in agile architecting." pp. 4807-4816.

[7] Z. Li, P. Liang, and P. Avgeriou, "Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios." pp. 65-74.

[8] T. Besker, A. Martini, and J. Bosch, “Managing architectural technical debt: A unified model and systematic literature review,” Journal of Systems and Software, vol. 135, no. Supplement C, pp. 1-16, 2018/01/01/, 2018.

[9] C. Seaman et al., "Using technical debt data in decision making: Potential decision approaches." pp. 45-48.

[10] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model." pp. 85-92.

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

[12] P. Avgeriou, P. Kruchten, I. Ozkaya, and C. Seaman, “Managing Technical Debt in Software Engineering (Dagstuhl Seminar 16162),” Dagstuhl Reports, vol. 6, no. 4, pp. 110-138, 2016.

[13] B. Curtis, J. Sappidi, and A. Szynkarski, “Estimating the size, cost, and types of technical debt,” in Proceedings of the Third International Workshop on Managing Technical Debt, Zurich, Switzerland, 2012, pp. 49-53.

[14] N. Mellegård, “Using weekly open defect reports as an indicator for software process efficiency: theoretical framework and a longitudinal automotive industrial case study,” in Proceedings of the 27th International Workshop on Software Measurement and 12th International Conference on Software Process and Product Measurement, Gothenburg, Sweden, 2017, pp. 170-175.

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

[16] V. S. Fonseca, M. P. Barcellos, and R. de Almeida Falbo, “An ontology-based approach for integrating tools supporting the software measurement process,” Science of Computer Programming, vol. 135, pp. 20-44, 2017/02/15/, 2017.

[17] M. Murray, and N. Kujundzic, Critical Reflection : A Textbook for Critical Thinking, Montreal, CANADA: MQUP, 2005.

[18] T. Besker, A. Martini, and J. Bosch, “Technical Debt Cripples Software Developer Productivity - A longitudinal study on developers’ daily software development work,” First International Conference on Technical Debt @ ICSE18, 2018.

[19] V. Basili, G. Caldiera, and D. Rombach, “The Goal Question Metric Approach,” Encyclopedia of Software Eng., J.J. Marciniak, ed., pp. 528-532, 19924.

[20] E. Oliveira, D. Viana, M. Cristo, T. Conte, and "How have Software Engineering Researchers been Measuring Software Productivity? A Systematic Mapping Study ". pp. 76-87.

[21] W. Scacchi, “Understanding Software Productivity, ,” International Journal of Software Engineering and Knowledge Engineering, vol. 1, no. 3, pp. 293-321, 1991.

[22] K. D. Maxwell, “Collecting data for comparability: benchmarking software development productivity,” IEEE Software, vol. 18, no. 5, pp. 22-25, 2001.

[23] R. W. Jensen, Improving Software Development Productivity: Effective Leadership and Quantitative Methods in Software Management: Prentice Hall Press, 2014.

[24] T. Sedano, P. Ralph, and C. e. Péraire, “Software development waste,” in Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, Argentina, 2017, pp. 130-140.

[25] 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 2015 10th Joint Meeting on Foundations of Software Engineering, Bergamo, Italy, 2015, pp. 50-60.

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

[27] R. Kazman et al., "A Case Study in Locating the Architectural Roots of Technical Debt." pp. 179-188.

[28] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt - When and by whom will it be paid? ." pp. 13-23.

[29] A. Martini, T. Besker, and J. Bosch, “Technical debt tracking: Current state of practice a survey and multiple case study in 15 large organizations,” Science of Computer Programming, 2018.

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

[31] R. J. Eisenberg, “A threshold based approach to technical debt,” SIGSOFT Softw. Eng. Notes, vol. 37, no. 2, pp. 1-6, 2012.
[32] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” Journal of Systems and Software, vol. 101, pp. 193-220, 2015.
[33] N. S. R. Alves et al., “Identification and Management of Technical Debt: A Systematic Mapping Study,” Information and Software Technology, 2015.
[34] P. Avgeriou, P. Kruchten, R. L. Nord, I. Ozkaya, and C. Seaman, “Reducing friction in software development,” IEEE Software, vol. 33, no. 1, pp. 66-72, 2016.
[35] P. Runeson, and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Software Engineering, vol. 14, no. 2, pp. 131-164, 2009.
[36] J. Miller, “Triangulation as a basis for knowledge discovery in software engineering,” Empirical Software Engineering, vol. 13, no. 2, pp. 223-228, 2008.
[37] S. McConnell. "Technical Debt. 10x Software Development [cited 2010 June 14]," http://www.construx.com/10x_Software_Development/Technical_Debt/.
[38] C. Wohlin et al., Experimentation in software engineering: an introduction: Kluwer Academic Publishers, 2000.
[39] B. A. Kitchenham et al., “Preliminary guidelines for empirical research in software engineering,” Software Engineering, IEEE Transactions on, vol. 28, no. 8, pp. 721-734, 2002.
[40] N. Juristo, A. M. Moreno, SpringerLink, and A. SpringerLink, Basics of Software Engineering Experimentation, 1 ed., Boston, MA: Springer US, 2001.
[41] D. F. Morrison, “The optimal spacing of repeated measurements,” Biometrics, vol. 26:281-90, 1970.
[42] R. Core-Team, "R: A language and environment for statistical computing. R Foundation for Statistical Computing."
[43] T. Therneau, and B. Atkinson, “ rpart: Recursive Partitioning and Regression Trees. R package version 4.1-13.
URL https://CRAN.R-project.org/package=rpart,” 2018.
[44] C. B. Seaman, “Qualitative methods in empirical studies of software engineering,” IEEE Transactions on Software Engineering, vol. 25, no. 4, pp. 557-572, 1999.
[45] R. K. Yin, Case study research: design and methods, London: SAGE, 2014.
[46] V. Braun, and V. Clarke, "Using thematic analysis in psychology, Qualitative research in psychology, 3(2)." pp. 77-101.
[47] 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.
[48] F. J. Shull, J. C. Carver, S. Vegas, and N. Juristo, “The role of replications in Empirical Software Engineering,” Empirical Software Engineering, vol. 13, no. 2, pp. 211-218, 2008/04/01, 2008.
[49] L. M. Pickard, B. A. Kitchenham, and P. W. Jones, “Combining empirical results in software engineering,” Information and Software Technology, vol. 40, no. 14, pp. 811-821, 1998/12/01/, 1998.
[50] M. Shepperd, N. Ajienka, and S. Counsell, “The role and value of replication in empirical software engineering results,” Information and Software Technology, vol. 99, pp. 120-132, 2018/07/01/, 2018.
[51] J. C. Carver, "Towards Reporting Guidelines for Experimental Replications: A Proposal."
[52] N. S. R. Alves, L. F. Ribeiro, V. Caires, T. S. Mendes, and R. O. Spinola, "Towards an Ontology of Terms on Technical Debt." pp. 1-7.
[53] V. R. Basili, F. Shull, and F. Lanubile, “Building knowledge through families of experiments,” IEEE Transactions on Software Engineering, vol. 25, no. 4, pp. 456-473, 1999.
[54] S. Vegas et al., “Analysis of the influence of communication between researchers on experiment replication,” in Proceedings of the 2006 ACM/IEEE international symposium on Empirical software engineering, Rio de Janeiro, Brazil, 2006, pp. 28-37.
[55] D. Graziotin, X. Wang, and P. Abrahamsson, “Do feelings matter? On the correlation of affects and the self-assessed productivity in software engineering,” J. Softw. Evol. Process, vol. 27, no. 7, pp. 467-487, 2015.

Terese Besker is a PhD candidate in the Software Engineering at Chalmers University of Technology, Sweden. She is working in the research fields of technical debt management. Before becoming a PhD student, she had worked as a senior software engineer in software industry for more than fifteen years. She also has a bachelor degree in software engineering and master degree in applied IT from Chalmers. She has published several peer-reviewed articles in journals, conference and workshop proceedings.

Antonio Martini is Associate Professor at the research group for programming and software engineering at the University of Oslo in Norway. Antonio has worked as a developer in industry, carrying out various roles. Antonio is leading a project within a research consortium of academia and large international companies on the management of Technical Debt. Antonio also works with software architecture, agile software development and software measurements.

Jan Bosch is professor of software engineering and director of the Software Center (www.software-center.se) at Chalmers University Technology in Sweden. Earlier, he worked as Vice President Engineering Process at Intuit Inc where he also led Intuit's Open Innovation efforts and headed the central mobile technologies team. Before Intuit, he was head of the Software and Application Technologies Laboratory at Nokia Research Center, Finland. Prior to joining Nokia, he headed the software engineering research group at the University of Groningen, The Netherlands. He received a MSc degree from the University of Twente, The Netherlands, and a PhD degree from Lund University, Sweden. His research activities include evidence-based development, software architecture, innovation experiment systems, compositional software engineering, software ecosystems, software product families and software variability management.