The Pricey Bill of Technical Debt

- When and by whom will it be paid?

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

Antonio Martini
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
antonio.martini@chalmers.se

Jan Bosch
Computer Science and Engineering
Chalmers University of Technology
Gothenburg, Sweden
jan.bosch@chalmers.se

Abstract— **Software companies need to support continuous and fast delivery of customer value both in short and a long-term perspective. However, this can be hindered by evolution limitations and high maintenance efforts due to internal software quality issues by what is described as Technical Debt. Although significant theoretical work has been undertaken to describe the negative effects of Technical Debt, these studies tend to have a weak empirical basis and often lack quantitative data. The aim of this study is to estimate wasted time, caused by the Technical Debt interest during the software life-cycle. This study also investigates how practitioners perceive and estimate the impact of the negative consequences due to Technical Debt during the software development process. This paper reports the results of both an online web-survey provided quantitative data from 258 participants and follow-up interviews with 32 industrial software practitioners. The importance and originality of this study contributes and provides novel insights into the research on Technical Debt by quantifying the perceived interest and the negative effects it has on the software development life-cycle. The findings show that on average, 36 % of all development time is estimated to be wasted due to Technical Debt; Complex Architectural Design and Requirement Technical Debt generates most negative effect; and that most time is wasted on understanding and/or measuring the Technical Debt. Moreover, the analysis of the professional roles and the age of the software system in the survey revealed that different roles are affected differently and that the consequences of Technical Debt are also influenced by the age of the software system.**

Keywords-component; Technical Debt, Wasted time, Development Cost, Software Development, Empirical Study, Survey, Qualitative data, Quantitative data

I. INTRODUCTION

Technical Debt (TD) is recognized as a critical issue in today's software development industry [33]. Since this phenomenon is detrimental to software companies, it is important to assess and estimate the consequences of Technical Debt in terms of the scale of wasted time and effort.

Ward Cunningham [7] introduced the financial metaphor of TD in order to describe to nontechnical product stakeholders the need to recognize the potential long-term negative effects of the immature code that is made during the software development life-cycle. Such debt has to be repaid with interest in the long term. Interest is the negative effect of the extra effort that has to be paid due to the accumulated amount of TD in the system, such as executing manual processes that could potentially be automated or spending excessive effort on modifying an unnecessarily complex code, performance problems due to lower resource usage caused by an inefficient code and similar costs [33],[8]. Ampatzoglou et al. [1] define interest as "*The additional effort that is needed to be spent on maintaining the software because of its decayed design-time quality.*"

Nowadays, it is well established that TD has a negative impact on software development organizations by hindering evolution and causing high maintenance costs due to internal software quality issues [17],[29],[20],[12],[21].

Left unmanaged, TD can result in unexpectedly large cost overruns, severe quality issues, inability to add new features [32] and even lead to a crisis point when a huge, costly refactoring or the replacement of the entire system needs to be undertaken [24],[5].

There are many different types of TD (such as Architectural TD, Requirement TD, Test TD, Code TD), which differ in the degree of their negative impact and consequently cause various levels of wasted time during the software development process. Several professional roles participate in the software development process, and could possibly be affected by TD in diverse ways. Furthermore, as the software ages, different types of TD could have varying negative effects and could possibly generate a dissimilar distribution of extra time spent on different activities.

However, little knowledge and few supporting tools are available to measure the extent of TD within a system and, in addition, the time spent on TD related issues is not made explicitly visible and measurable. Without such knowledge, software development organizations do not know the interest that they are paying on the debt, and therefore they might not give TD management the necessary attention.

We will answer the research questions by using survey data based on software professionals' perceptions. All survey respondents were experienced in software development, and therefore, their estimates were likely to be formed by what they have heard, observed, and experienced at their workplaces.

To the best of our knowledge, there are no studies quantifying the interest in terms of how much time (observed, measured or estimated) is wasted due to TD and how this wasted time varies in relation to the system age and its impact on a range of professional software roles.

We, therefore, aim to answer the following research questions (RQ):

RQ1. *How much of the overall development time is wasted because of Technical Debt?*

RQ2. *What kind of Technical Debt Challenges generates the most negative impact?*

RQ3. *What are the various activities on which extra-time is spent as a result of Technical Debt?*

RQ4. *In what way does the age of the software system affect the questions stated in RQ1-3?*

RQ5. *In what way are the different professional software roles affected by the questions stated in RQ1-3?*

This study offers a contribution to research, with respect to the existing body of knowledge about TD in the following important areas:

1) We provide a survey- and interview-base study on how practitioners experience TD within the software industry based on both quantitative and qualitative data.

2) This study attempted to quantify the interest in terms of wasted software development time. We present results that the average *estimated wasted software development time is 36%,* because of TD.

3) We present results showing that Complex Architectural Design, closely followed by Requirement TD, generates the most negative impact on daily software development work.

4) This study provides new insights into TD research by showing that the most wasted time is spent on *Understanding and/or Measuring* the TD.

5) This research reveals that different roles within software development are affected differently by TD. We found that the amount of estimated time spent as interest of TD is supported by all roles. Furthermore, different roles waste time on different activities, hence experiencing different negative impacts of TD.

6) This study shows that the degree of wasted time varies with the age of the software. Although the amount of wasted time result does not show a linear progression, the wasted time varies in relation to the system age.

7) To both practitioners and academics, this study demonstrates the relevance of paying more attention and effort to remediate TD deliberately.

The remainder of this paper is structured in seven sections, where the second section introduces related work. In the third section, the research method is described. The fourth section presents the results that are discussed in section five. Finally, in section six, threats to validity are presented, and section seven concludes the paper.

II. RELATED WORK

This section presents related research, both based on empirical survey-based data and on other TD related studies.

1) Empirical survey-based studies on TD

By reviewing other survey-based studies focusing on TD, we found four survey-based studies which somewhat related to our investigation.

Ernst et al. [10] conducted a survey within three large organizations, with 536 respondents and seven follow-up interviews. The result of that study shows that "bad architecture choices" are the greatest source of TD. This study also concludes that the degree of architectural drift is related to system age. They found a weak association between system age and the perceived importance of architectural issues where 89% of the respondents with system age $\geq$ 6 years agreed or strongly agreed with the notion that architectural issues were a significant source of debt, compared to 80% of those with newer. In contrast, to that study, our study also includes how different *roles* perceive TD and also has a finer granularity of the age intervals and therefore, provides a more detailed analysis. Furthermore, this study provides a wider coverage in terms of including an investigation of how ***all*** different TD issues, not only the architectural related TD, vary in relation to the system age.

Holvitie, Leppanen and Hyrynsalmi [14] conducted a survey of 54 Finnish software practitioners investigating the level of TD knowledge, how TD occurs in projects and which of the applied agile components of the development was sensitive to TD. They found that the most frequent causes of TD were inadequate architecture and inadequate documentation. In this study, we cannot find a quantification or estimation of the interest, there is no explanation about what type of activities on which extra-time is spent, and perspective of different roles or age of the software are not investigated.

This paper is somewhat also related to our previous papers [3],[5], where we specifically study the *architectural type of* TD and address how TD negatively affects different *software quality* attributes. However, even though the data were collected using the same survey, this study uses different data and focusing on several different types of TD and compare those with each other regarding the scale of wasted time and effort, rather than quality.

2) Prior research on TD related issues

Kazman et al. [15] present a case study of identifying and quantifying architectural debts in an industrial software project, using code changes as a proxy for calculating the interest. This study focus on identifying the architectural roots of TD, meaning that the study does not address the cost of interest, but instead aims at quantifying the expected payback for refactoring. 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. Compared to that study, this study focus on several different types of TD (not only on the architectural TD) and this study have a focus on quantifying the interest in terms of wasted time instead of a focus on locating the roots of the architectural TD and the expected benefit from the refactoring. Falessi et al. [11] argue that the interest is non-linear and has a probability to grow exponentially rather than linearly. By analyzing source code, Nugroho et al. [28] describe that the interest grows differently on a 10-years horizon, depending on a star rating system of the software. They present calculations based on assumed values and empirical data which illustrates that the interest grows linear for 3-star systems and close to linear for 4-star systems. However, such study is based on known metrics (which do not cover the full spectrum of TD types) and on code changes as an estimation of interest.

In summary, although, there have been some attempts on quantifying the interest of TD, there is a lack of empirical investigations including software practitioners. Also, TD interest has not been studied holistically (including all types of TD). Finally, the interest has not been put in relation to the system age and to how it affects different professional roles.

III. METHODOLOGY

In this study, a combination of quantitative and qualitative research approaches is used. We aimed at using methodological, source and observer triangulation in order to increase the validity and the reliability of the result. Methodological triangulation refers to the utilization of both qualitative and quantitative methods for gathering data [26], source triangulation refers to using different sources of data while observer triangulation refers using more than one observer in the study [30]. Triangulation is important to increase the precision of empirical research and thus to provide a broader picture [30]. The following sections describe the methods used for conducting the survey and the complementing qualitative follow-up interviews.

A. Survey

The web-survey was designed and hosted by the online survey service called *SurveyMonkey*. The survey was using a mix of open- and closed-ended questions. The questions were a combination of optional and mandatory nature. To avoid bias in the survey, the questions were developed as neutral as possible, ordered in a way that one question did not influence the response to the next question, and a description was provided when needed [16]. The first draft of the survey was tested by four industrial practitioners (developer, manager, project owner, and software architect) and by two Ph.D. candidates in order to evaluate the understanding of the questions and the usage of common terms and expressions [9]. During this evaluation, we also monitored the time needed to complete the survey. The survey was made accessible between February and March 2016, and a reminder was sent out after two weeks to those who had been specifically invited. The survey was anonymous, and participation in the survey was voluntary.

1) Data collection

The survey invitation was mailed directly to seven companies/partners within our networks, all located in Scandinavia, with an extensive range of software development, and invitations were also published at software engineering related networks on LinkedIn. Across all these collaborators, 312 respondents began the survey, and 258 respondents answered all questions. Due to this high completion rate (83%), we decided to reject the incomplete questionnaires, according to the guidelines by Kitchenham and Pfleeger [16].

The first part of the survey gathered descriptive statistics to summarize the backgrounds of the respondents and their companies. This data is compiled and presented in Table I. The level of education of the respondents was quite high, with 58% having a Master degree and 25% having a Bachelor degree. The survey included respondents having different roles, where 49% were Developers/Programmers/-Software Engineers, while 25% were Software Architects. Approximately 78% of the Software architects and 59% of the Developers/- Programmers/Software Engineers had more than ten years of experience. The most common size of the software development team included 6-10 members (36 %), and most (32%) systems were on average 5-10 years old from their initial design. Nevertheless, a significant number (35%) of the respondents' systems were more than ten years old. Most of the software systems were embedded systems (49%) and Real-time system (42%). However, in this specific survey question, the respondents could select more than one option.

The second part of the survey included questions based on the research questions presented in Section I.

In this part of the survey, the participants were asked to estimate their perception of the three survey questions (SQ):

SQ1. Which of the following *challenges* generates the most negative impact on your daily software development work? Please rank them from 1 to 11.

SQ2. How much of the overall development time is wasted because of these issues? (Do not consider fixing and managing the issues, just if they hinder you when you add/change/understand the system).

SQ3. If you take into consideration your most recent projects and the issues listed in SQ1, which of the following activities did you spend most of your extra time on? Please rank them from 1 to 6.

TABLE I. CHARACTERISTICS OF THE SAMPLE SURVEY

Individual		Company	
Experience		**Team size**	
< 2 years	3,90%	1-5 members	23,30%
2 - 5 year	10,50%	6-10 members	36,00%
5 - 10 year	17,40%	11-20 members	15,90%
> 10 years	68,20%	21-40 members	6,60%
Education		> 40 members	18,20%
Master degree	57,80%	**Software system type***	
Bachelor degree	25,20%	Embedded system	48.84%
No Univ. education	6,20%	Real-time system	41.86%
Other:	5,80%	Data management system	22.09%
Ph.D. degree	5,00%	System Integration	20.93%
Roles		Modeling and/or simul.	15.12%
Developer/Program/-Software Engineer **	49,20%	Data analysis system	14.73%
Software Architect	24,80%	Web 2.0 / SaaS system	8.53%
Manager	6,20%	Other	8.53%
Project Manager	6,20%	**System Age**	
Product Manager	5,0%	< 2 years	9,70%
Expert	5,0%	2-5 years	23,3%
Tester	3,50%	5-10 years	32,2%
Gender		10-20 years	28,3%
Male	89,90%	>20 years	6,6%
Female	8,50%		
Other/no share	1,60%		

* More than one option were selectable, ** Abbreviated as ***Developer*** in this paper

For the question SQ1, the different categories and sub-categories provided by [19] were used to distinguish the different TD types (Complex Architectural Design, Requirement TD, Testing TD, Source Code TD, Documentation TD, Too many different patterns and policies, Dependency violations, Infrastructure TD, Lack of reusability in design, Dependencies to external resources/software, and Uneasy/Tensed social interactions between different stakeholders).

In survey questions, SQ2 and SQ3, these same types of TD were referred to in order to ensure a better construct validity of the survey. In SQ2, the estimation of the interest as wasted time was indicated in predefined percentage intervals of <10%, 10-20%....80-90% and I don't know. In order to separate the time spent on management, the respondents were asked not to include the time spent on fixing and managing the issues. In SQ3, the respondents could indicate their level of agreement on a 6-point Likert Scale (Strongly agree …Strongly disagree).

2) Data analysis

The data from the survey was analyzed in a quantitative fashion, i.e. by interpreting the numbers obtained from the answers. All analyses were carried out using the software SPSS (version 22). Descriptive statistics were employed to organize and analyze the qualitative data by using tables and charts. To allow for ordinal comparison, a transforming from Likert scale to numeric value has been used. The assigned ordinal values for the Likert scale categories were: Never = 1, Rarely = 2, Occasionally = 3, Frequently = 4 and Very Frequently = 5.

The data was analyzed by studying the median, mean and standard deviation and also by using the statistical methods Pearson's R, the Pearson chi-square tests, and ANOVA. The Pearson's R method computes the pairwise determining the strength and direction of the association between two metrics and can be used to measure a linear association between two metrics. The Pearson chi-squared tests are used for evaluating how likely it is that any observed difference between the sets arose by chance, and the test of independence assesses whether unpaired observations on two variables are independent of each other. The one-way ANOVA was used to identify significant differences in mean values. Mainly three different types of quantitative analysis were conducted, in order to synthesize the result for each research question:

- **General results:** all the answers are considered, composed and used as a result to draw conclusions common to all roles and all system ages.
- **System Age:** all answers are grouped into system age intervals of <2 years, 2-5 years, 5-10 years, 10-20 years, and >20 years. The time refers to the age from initial design. The age intervals are treated as categorical data.
- **Role:** all the answers are grouped into roles of Developer, Expert, Manager, Product Manager, Project Manager, Software Architect, and Tester.

B. Interviews

1) Data collection

Interviews can be divided into unstructured, semi-structured and fully structured interviews. As suggested in [31], this study employed the technique of semi-structured interviews where the questions were planned but not necessarily asked in the same order as they were listed. This semi-structured interview technique allowed flexibility and exploration of the studied objects by asking follow-up questions based on the respondents' answers. All interviews were group interviews and were conducted using the guidelines suggested by Krueger and Casey [18]. In total, we interviewed seven companies, where each interview included between 4 to 7 participants. These companies did all participate in the survey, and all the interviewees had answered the survey before the interview. Altogether, we interviewed 32 experienced software development professionals with roles as architects, developers, product owners and managers. Each interview lasted between 105 and 120 minutes and was digitally recorded and transcribed verbatim. During the interviews, compiled results from the previous survey were presented to the respondents, where some of the most interesting findings were highlighted together with questions related to the specific area of research. This presentation allowed the respondents to more easily relate the interview questions to the result of the survey. The interview questions were designed to a) increase the understanding of the survey results, b) ensure that the questions in the survey were understood and interpreted as intended and in a uniform way, c) confirm the result from the survey, and d) understand the implications of the survey

results. The questions were developed to cover the same taxonomies as in the survey. The following are examples of the questions asked during the interview:

- What do you consider to be a *Complex Architectural Design*?
- If the wasted time could be reduced, how would that affect your software development process?

2) Data analysis

The transcriptions from the recorded interviews were manually coded using established guidelines in the literature [2]. To assess and rate the level of confirmation by the interviewees of the survey result, a three-level classification of the confirmation was used; *Strongly Confirming* (SC), *SomeWhat Confirming* (SWC) and *DisConfirming* (DC). The result from the survey, which was expressly and unambiguously confirmed by interviewees, was classified as *SC*. The results which were not confirmed or those that were confirmed but with deviations were classified as *DC*.

IV. Results and Findings

The following subsections present results for the research questions presented in Section I, and the results are grouped according to each research question.

1) How much of the overall development time is wasted because of Technical Debt? (RQ1).

This research question focuses on the TD interest in terms of how much of the overall development time is wasted. The estimated wasted time does not include fixing and managing the issues. Fig. 1 provides an overview of the distribution of the estimated wasted time which is represented by different percentage intervals of the overall development time. The most striking result emerging from the data is that, on average, the respondents estimated that **36% of all software development time is wasted because of paying the interest of TD** (excluding the Don't know option -3,5% of the respondents), with the standard deviation of 17,8%. It can be seen from the data in Table II that the result from the survey was almost consistently *Strongly Confirmed* (SC) during the group interviews. Only during one interview (Intv 6), one interviewee expressed "*It is probably more than 36%* ".

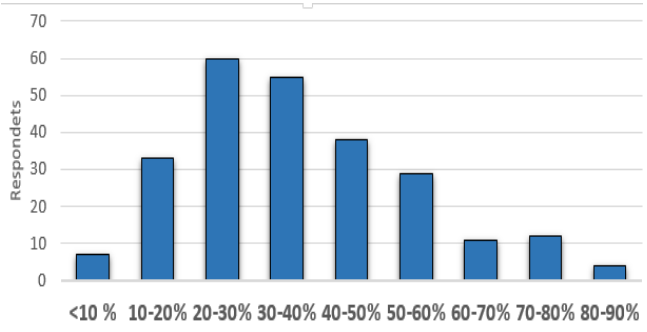

Figure 1. Estimated wasted software development time

Furthermore, the interviews also coherently revealed that if this detrimental waste of time could be reduced, the company would spend more time on adding new features and reducing the time to market for their software products. The interviewees also unanimously described that this huge waste of time is detrimental to their work, and its implications must be clearly recognized and understood.

TABLE II. Interview Confirmation – Wasted time

Intv 1	*Intv 2*	*Intv 3*	*Intv 4*	*Intv 5*	*Intv 6*	*Intv 7*
SC	SC	SC	SC	SC	SWC	SC

2) System Age effect on wasted time (RQ4)

In order to assess whether the interest changed or not during the software life cycle, a statistical cross-tabulated analysis showing the interest in terms of the wasted time in relation to the system age interval was used. Fig. 2 captures the difference between the estimated time wasted in relation to the age of the software. For a system with age less than two years, on average, 33.8% of the software time is estimated to be wasted but for a system older than 20 years, the estimated wasted time average is 40.5%. The degree of the wasted time thus varies with the age of the software and to evaluate if there is a linear correlation between the system age and the wasted time, we used the Pearson correlation method.

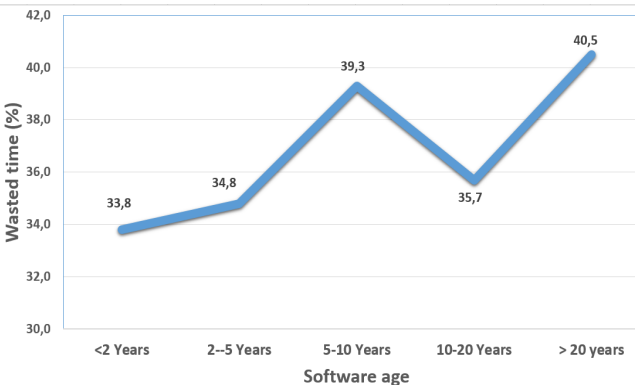

Figure 2. Wasted time in relation to System Age

These two data series (we used the average age within each year interval), returned a p-value of 0.059, which showed no support for a linear correlation of the wasted time and the age of the system. We concluded that no evidence was found for linear associations between the amount of wasted time and the age of the system.

Furthermore, the Pearson Chi-square test of dependence ($\chi 2$= 58.64, df=36, p=0.010) showed that the two factors (wasted time and system age) are not independent, which means that we could not reject the hypothesis that the system age influences the wasted time.

3) Different Roles' estimation of the wasted time (RQ5)

In this section, we explore whether different professional roles estimate the wasted time differently. When tested

explicitly if the roles estimate the wasted time differently, a one-way ANOVA test showed that there were no significant differences (F(6.242)=0.926,p=0.477).

To graphically visualize the differences of distributions of the estimated wasted time, Fig. 3 represents a boxplot for each of the roles. The box-plot represents the minimum and maximum range values, the upper and lower quartiles, and the median. From the boxplot, we can see that the estimated wasted time is relatively consistent with the different roles (median range of 30-45 % and standard deviation range of 14.4–20.6 %). In this figure, we can also see that Software Architects and Experts estimate the highest average percent of the wasted time (41%), whereas Developers estimate the lowest value of wasted time (35%). One unanticipated result of the analysis showed that five of the Software Architects and seven of the Developers were found to estimate their wasted time to more than 70%.

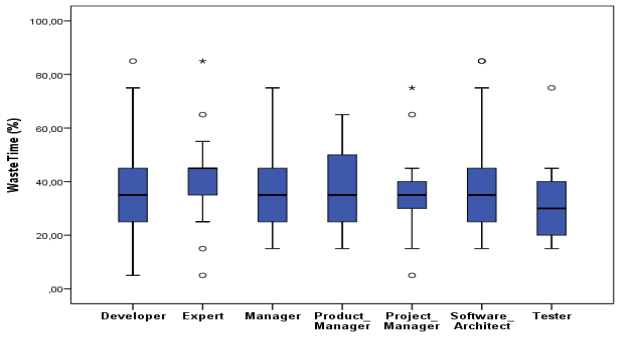

Figure 3. Distribution of the estimated Wasted Time by different Roles

A. *What Challenges generate the most negative impact on the daily software development work? (RQ2)*

To understand what type of TD generates the most negative effect, the respondents were asked to rank (score 1 = lowest impact, score 11 = highest impact) a randomly ordered list of 11 different *challenges* which they consider to have the most negative impact on their daily software development work. The different listed *challenges* reflected the different types of TD that emerged from [19].

There are several possible statistic values to study when evaluating a ranking result. This question involves an examination of the mean, the median and a grouping of different sets of scores.

In Fig. 4, the summary statistics for each Challenge reveal that a *Complex Architectural Design* (mean rank 7.27) and *Requirement TD* (mean rank 7.23) generates the most negative impact on daily software development work, both when studying the mean and the median values. To make it easier to understand, measure, analyze and compare the different *challenges*, the ranking scores were broken down into smaller subintervals (called class intervals), where the highest scores 9-11 are aggregated as *High*, score 4-8 are aggregated as *Medium* and scores below 4 are aggregated as *Low*.

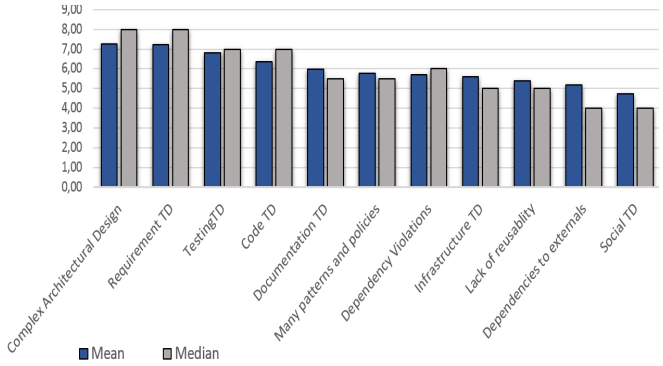

Figure 4. Mean and median of the rankings for each Challenge

For each class interval, the amount of data items falling within each impact interval is counted.

In Table III, the frequency for each *challenge* is tabulated and compared by the aggregations of three different levels of negative impacts on the daily software development work. To make it easier when comparing the values, the cells in the table are colored (the grayer the cell, the higher the frequency is). As it can be seen from this table *Complex Architecture Design* (42.2%), *Requirement TD* (40.3%), and *Testing TD* (37.6%) reported significantly increased frequencies in the highest impact interval, compared to the other listed *challenges*.

As illustrated in Table IV, during the group interviews where this result was presented, the interviewees *Strongly Confirmed* (SC) this result as reflecting their interpretation of the *challenges* within their organizations consistently.

TABLE III. FREQUENCY OF CHALLENGES

Challenge	Frequency (%)		
	High	***Medium***	***Low***
Complex Architectural Design	**42,2**	**38,0**	**19,8**
Requirement TD	**40,3**	**38,4**	**21,3**
Testing TD	**37,6**	**34,1**	**28,3**
Source Code TD	**29,1**	**38,4**	**32,6**
Infrastructure TD	**25,2**	**29,8**	**45,0**
Documentation TD	**24,8**	**41,9**	**33,3**
Dependencies to external resources/software	**22,5**	**26,7**	**50,8**
Too many different patterns and policies	**21,3**	**41,1**	**37,6**
Uneasy/Tensed social interactions between different stakeholders	**19,4**	**24,8**	**55,8**
Lack of reusability in design	**19,0**	**39,1**	**41,9**
Dependency violations	**18,6**	**47,7**	**33,7**

TABLE IV. INTERVIEW CONFIRMATION – CHALLENGES

Intv 1	*Intv 2*	*Intv 3*	*Intv 4*	*Intv 5*	*Intv 6*	*Intv 7*
SC	SC	SC	SC	SC	SC	SC

1) System Age effect different negative effects. (RQ4)

To assess whether the age of the system has an influence on the *challenges* during the software life-cycle, we measured the pairwise combinations for the different system age intervals with the means of the *challenges*.

Fig. 6 shows each *challenge's* impact in relation to the system age interval, that respondents of different software system ages interpret the *challenges* differently and that the impact of the *challenges* varies over time.

Determined by one-way ANOVA, there is a statistically significant difference between the mean value in the system age intervals for *Requirement TD* ($F(10)= 1.917$, $p= .043$), *Complex Architectural Design* ($F(10)= 1.928$, $p= .042$) and *Uneasy/-Tensed social interaction*s ($F(10)= 1.911$, $p= .044$). Furthermore, the data in Fig. 6 reveals that Requirement has the highest mean value for systems that are less than ten years old, whereas, for systems older than 10 years, a Complex Architectural Design has the highest mean value.

To evaluate if there are any correlations between the time intervals (where the average age within each interval is used) and the median of the scores, we calculated the Pearson correlation coefficient by measuring the strength of a linear relationship between the pair data. The Pearson correlation coefficient for all the data series varied between 0.013 and 0,179 (independent of a positive or negative correlation). The test is not indicating any significant linear correlations in any time interval. Supplementary, a Pearson chi-square test for independence of each challenge was calculated to evaluate if any factors gave a significant response (p-value < 0.1). This test showed that the factors for system age and Requirement ($\chi2= 59.33$, $df= 40$, $p= 0.025$), Patterns and policies ($\chi2= 58.8$, $df = 40$, $p = 0.028$), and Social interactions ($\chi2= 77.5$, $df = 40$, $p = 0.000$) are not independent. This indicates that there is a significant relationship between these *challenges* and the system age.

The previous result showed that *Complex Architectural Design* and *Requirement TD* generate the most negative impact on daily software development work, and the value of *Requirement TD* was observed having the absolute highest negative impact within the age interval of 2-5 years (mean value 7.87). The value of *Complex Architecture Design* was observed having the highest negative impact on systems with an age interval of 10-20 years (mean value 8.08) and the lowest value for the age interval of <2 years (mean value 6.56).

2) How different Roles interpret the Challenge (RQ6)

In this section, we explore how different roles interpret which Challenge has the *most*, and *second most* negative impact on their daily software work, by a cross-tabulation of the two data sets. Table V provides the mean value and the highest and second highest rates (aggregation of score 9-11) of the *challenges* for each role, using the same aggregation as described in Section IV.B. Table V is quite revealing in several ways; it is apparent that for Developers, Product Managers, Project Managers, and Testers, the *Complex Architecture Design* has the most negative impact. Meanwhile, Software Architects and Managers interpret the *Requirement TD* to generate the most negative impact.

A striking result emerging from Table V is also that *Tester* is the role that estimates the greatest negative impact of *Complex Architectural Design*, were 56% or these professionals estimates this *challenge* is having the highest rankings (scores >8) with a mean value of 8.1. The overall result shows that *Complex Architecture Design* is the most commonly estimated challenge by having the greatest negative impact on the daily software development work among all the different roles.

However, it is also worth noting that both *Requirement TD* and *Testing TD* are frequently estimated as having the second most negative impact on respondents' daily software work.

TABLE V. ROLES INTERPRET NEGATIVE EFFECT OF CHALLENGES

Role	**Negative impacts**		
	Challenge (first and second rate)	***High (%)***	***Mean Value***
Developers	1.Complex Architectural Design 2.Testing and Requirement	43 39	7,4 6,9/7,2
Experts	1.Documentation 2.Requirement	54 46	7,1 7,2
Managers	1.Requirement and Testing 2. Complex Architectural Design and Lack of reusability	50 44	8,6/7,2 7,3/6,9
Product Managers	1.Complex Architectural Design 2. Environment and infrastructure and Lack of reusability	46 38	7,6 6,5/6,5
Project Managers	1.Complex Architectural Design 2. Testing and Requirement	50 44	8,3 7,0/7,9
Software Architects	1.Requirement TD 2. Low Code quality	44 41	7,3 6,9
Testers	1.Complex Architectural Design / to many Patterns and policies 2. Environment and infrastructure	56 44	8,1/7,2 6,9

B. What are the various activities on which extra-time is spent as a result of Technical Debt ? (RQ3)

Due to TD in software systems, a lot of extra time is spent on different activities which do not deliver any direct value to the customer. In an attempt to understand how this extra time is spent, we asked the respondents to the survey to refer to their most recent project and rank different listed activities on which they spent their most extra time.

The ranking scale was set from 1 to 6, where score 1 refers to that activity of which the *least* extra time is spent while score 6 represents the activity of which *most* extra time is spent. The specified activities were recognized during previous research by interviews with practitioners as common activities within the TD research field [23]. It can be seen from the data in Table VII, that the survey results showed how 20.63% of the respondents estimated that most extra time (score 6) is spent on *Understanding and/or measuring the issues* (mean value 4.38), while 22.98% of the

respondents estimate that most extra time (score 6) is spent on *Management processes* such as tracking, monitoring, and communication of the issues (mean value 3.89). Only 2.05% of the respondents pointed out that they spend most extra time on the activity of *Deciding, which issue to refactor first.*

As illustrated in Table VI, all the interviewees *Strongly Confirmed* (SC) the results from the survey on the spending of the extra-time.

TABLE VI. INTERVIEW CONFIRMATION – ACTIVITIES

Intv 1	*Intv 2*	*Intv 3*	*Intv 4*	*Intv 5*	*Intv 6*	*Intv 7*
SC	SC	SC	SC	SC	SC	SC

1) System Age effect how the extra time is spent (RQ4)

To investigate if and in what way the system age of the software influence how the extra time is spent, a cross-tabulated table was created with the mean value of each activity and the system age intervals. In addition, we used Pearson correlation analysis to test the null hypothesis that the interest for each activity grows linearly in relation to the age of the system. Even if the distribution is not normal distribution (due to Shapiro-Wilk test, $p>.05$), we have applied the statistical test Pearson R since this method is not very sensitive to moderate deviations from normality [22].

Results obtained from the survey are visualized in Fig. 5 and in Table VII. The examination of the highest mean value for software with a system age less than two years reveals that most extra time is spent on the activity of *Understanding and/or measuring* the issues (mean value 4.2). This activity slightly increases for the system age intervals of 2-5 years (mean value 4.3), 5-10 years (mean value 4.4) and 10-20 years (mean value 4.5) and finally, for software with a system age more than 20 years the time spent on this activity decreases back to the initial mean value 4.2.

From the boxplot in Fig. 5, we can see the distribution of the rating of the time spent on *Understanding and/or measuring the issues* in relation to the system age. It shows that the median of the rating changes between 4 and 5 and that the answers are unevenly distributed among the different system age intervals. A striking observation emerging from the data comparison was that the extra time spent on the *Deciding which issues to refactor first* activity is the lowermost in all system age intervals, with a mean value varying between 2.6 and 2.9.

In order to test the hypothesis that the time spent on these activities increase in relation to the system age, we studied if and how the time spent on the different activities changed in relation to the system age.

First, the Pearson correlation coefficient for the system age intervals where the average year within each interval is used. The variables for each activity varies between 0.012 and 0.041 (independent of positive or negative correlation), thus not indicating any significant linear correlation between the time spent on each activity in relation to the system age.

Secondly, results from a one-way ANOVA revealed no significant differences among the time spent on the activity for each system age interval. We, therefore, reject the hypothesis that the time spent on these activities significantly differs or linearly increases in relation to system age.

TABLE VII. EXTRA TIME SPENT ON ACTIVITIES

Activity	*Mean*	*Score 6 (highest)*
Understanding and/or measuring the issues	4.38	20.63 %
Management process of the issues (track, monitor, communicate)	3.89	22.98 %
Finding the issues	3.69	19.68 %
Refactoring the issues	3.59	15.35 %
Deciding which issue to refactor first	2.89	2.05 %
Other	2.50	15.15%

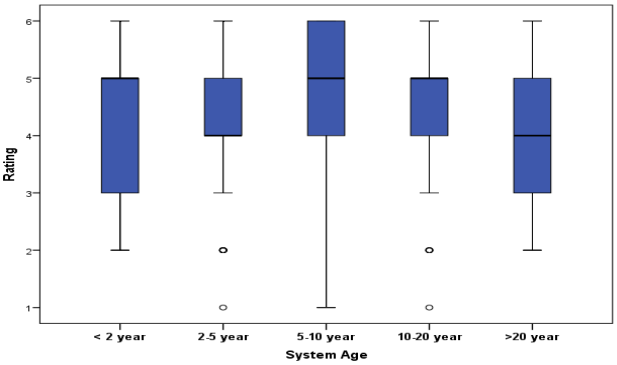

Figure 5. Distribution of understanding and/or measuring the issue

2) How professionals with different roles spend their extra time on different activities (RQ6)

It is inevitable that professionals having different roles will spend their extra time differently and on different activities. Accordingly, we wanted to explore how the roles estimate the extra time spent on the different activities. Fig. 8 compares the summary statistics on how the roles estimate the time spent on different activities, and it is evident from the figure that different roles spend their extra time on different activities.

We can see that Developers, Experts, Product Managers, and Project Managers spend most extra time on *Understanding and/or measuring the issues.* Meanwhile, Managers, Software Architects, and Testers spend most of their extra time on the *Management process.* Interestingly, the Managers represent the role that spends more extra time on the *Refactoring the issues,* and they are closely followed by Experts and Software Architects.

Further, determined by one-way ANOVA, is a statistically significant difference between the mean value describing how the roles spend the time for the activities of *Finding the issue* ($F(6)=2.201, p=.044$), *Understanding and/or measuring the issues* ($F(6)=2.894$, $p=.01$), *Management process of the issues* ($F(6)=4.336$, $p=.000$).

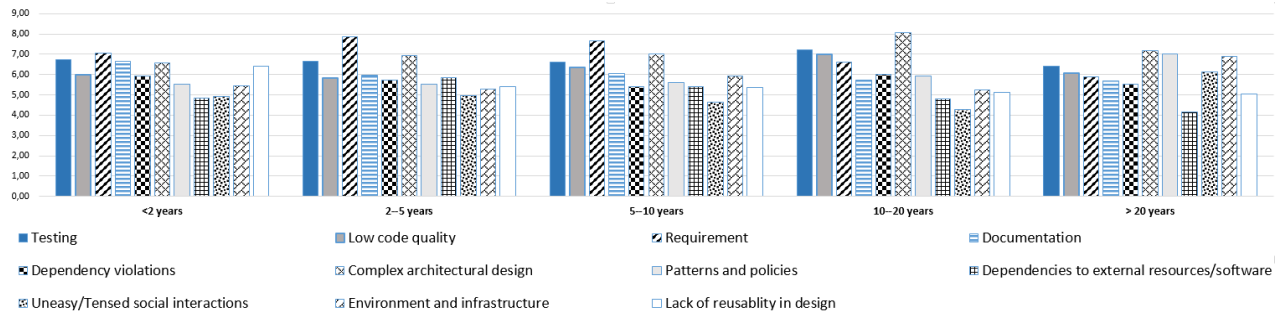

Figure 6. The mean value for each challenge in relation to system age

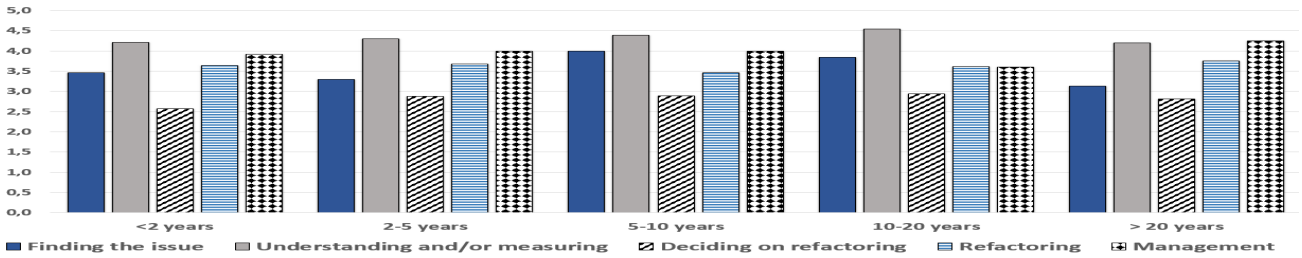

Figure 7. Extra time spent on different activities and system age

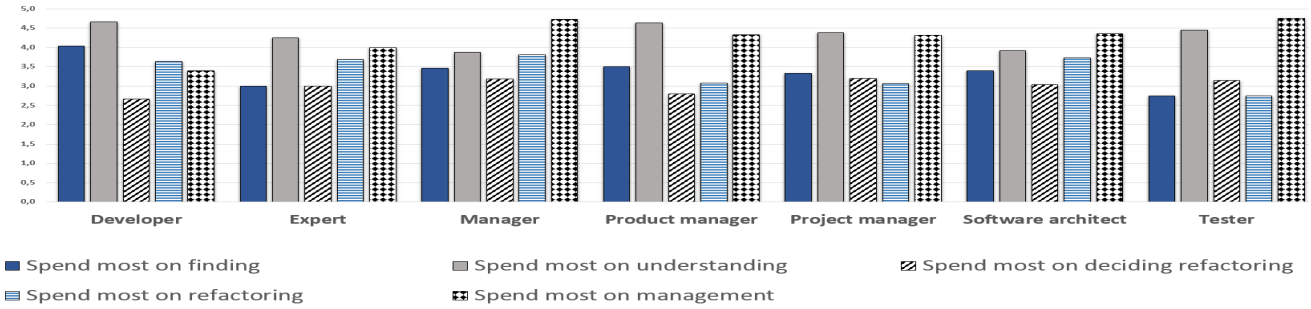

Figure 8. How professionals with different roles spend their extra time

V. DISCUSSION

1) Wasted time due to Technical Debt (RQ1, RQ4,RQ5)

The first research question (RQ1) aims to address how much of the overall development time is wasted because of TD. The striking results of the survey show that on average, the respondents estimate that **36%** of all software development time is wasted due to TD. The wasted time varies somewhat in relation to the system age but the distribution of the wasted time did not have a significant positive linear correlation with the system age (RQ4). However, the analysis from the survey shows that the most wasted time (40.5%) occurs in systems older than 20 years. Besides, even for new software systems with an age of less than two years, 33.8% of the time is estimated as wasted.

This result implies that TD is introduced early, and then it persists throughout the whole software life cycle.

The findings from the survey show that there is no significant difference in how the different roles estimate the overall wasted time (RQ6). However, the results show that Software Architects and Experts estimate the highest average value of wasted time (41%) and Developers estimate the lowest value of wasted time (35%). A possible explanation for these results may be that Software Architects have higher awareness about the complexity of the architectural design and, as identified in RQ2, a Complex Architectural Design is a common source of the most negative impact on daily software development work. Most of the interviewees explained that the magnitude of wasted time was consistent with their perception of the overall wasted time. They also claimed that this huge waste of time is detrimental to their business and its implications must be clearly addressed and recognized within their organizations.

2) What type of TD generates the most negative impact on daily development work (RQ2, RQ4, RQ5)

The second research question (RQ2) sought to determine what challenges generate the most negative impact on the

daily software development work, in order to understand and weight different TD issues by potential severity. The overall survey results reveal that *Complex Architectural Design,* closely followed by *Requirement TD,* generates the most negative impact on daily software development work where the interviewees described the characteristics of a complex architectural design as *"One part of this is when you are afraid to change something because something which is completely unrelated gets corrupted through that. Then it is complex"*. This finding broadly supports the work of other studies in this area by describing that TD instances linked to inadequacies in the software architecture are a major source of TD [14],[10], [6],[25]. Further, this study investigated the impact of the age of the software and if different roles experienced the *challenges* differently in the survey. According to these data, we can infer that the age of the system has a significant impact on *Architectural Complexity*, *Requirement TD and Social interactions* (RQ4). The impacts vary in relation to the system age (not linear), and *Complex Architecture Design* was observed having the highest negative impact on systems with an age interval of 10-20 years and the lowest value for the age interval of <2 years. The different role related findings report that Developers, Product and Project managers, and Testers rate the *Complex Architectural Design* as having the most negative impact (RQ5). Meanwhile, Software Architects interpret the *Requirement TD* to generate the most negative effect. These findings raise intriguing questions regarding if Software Architects are not prone enough to consider their own working area as the largest source of TD. However, another possible explanation for this could be that they actually have greater insight and understanding than others in this specific area. During the interviews, one of the interviewees commented on this result as "*Because software architects understand the architecture better and need requirements to create it, and thereby seeing the requirement as the biggest problem.*"

3) Extra time is spent on activities (RQ3, RQ5, RQ6)

The third research question (RQ3) focuses on how much extra time is spent on different activities due to the payment of TD interest. Both interviewees and survey respondents placed significant emphasis on describing that considerable time is spent on *Understanding and/or measuring* TD issues. This finding broadly supports the work of other studies in this area where the need for active TD management on especially architectural studies, is highlighted [27],[23],[4], [13]. Of interest is also the fact that *Understanding and/or measuring of Technical Debt* does not change due to the system age (RQ4), but is continuously at the highest level throughout the whole software life cycle. From the data, it is apparent that Developers, Experts, Product and Project Managers spend most extra time on *Understanding and/or measuring the issues*. Meanwhile, Managers, Software Architects, and Testers spend most of their extra time on the *Management process of the issues* (RQ6). This result could imply that different roles need different types of support.

VI. Threats to validity and Verifiability

For verifiability reasons, we have made information available online, to support a full or partial independent replication of the claimed contributions. All survey questions, used in this paper, are available on the link https://zenodo.org/record/437597#.WNOXw1PhCpo.

The qualitative data derived from the survey are not based on measured or observed data but on estimations made by the respondents. In future studies, we plan to include physical measurements and observations, to create a stronger reliability of the data. The result of this research may be affected by some threats to validity. *Construct validity* reflects what extent the operational measures that are studied represent, what the researchers have in mind and what is investigated according to the research questions [30]. To mitigate this risk and to make sure that the respondents were considering the correct type of TD issues, a short description of each type of TD was used and named as a *challenge*. Further, this study could possibly suffer from *internal validity* when causal relationships were examined as it affects our ability to explain the phenomena that we observed [34]. *External validity* focuses on to what extent it is possible to generalize the findings. There is always a risk in surveys that the sample is biased and for this topic, a potential threat refers to the demographic distribution of response samples. As reported in Section III.A.1, we mainly investigated companies from the Scandinavian area. To mitigate this validity issue, we attempted to enlarge the respondent's sample by inviting additional participants globally via LinkedIn. Without replicating this study to other countries, it is not possible to confirm that this study is generalizable. *Reliability* addresses whether the study would yield the same results if other researchers replicated it. To mitigate this threat, we have employed source triangulation, methodological triangulation and observer triangulation.

VII. Conclusion

This is the first study surveying the estimated magnitude of the interest paid on the accumulated TD in terms of perceived wasted time and effort. This study is based on a survey with 258 respondents and group interviews with 32 practitioners. The study has shown that software development practitioners estimate that *36 % of all development time is wasted* due to TD and that *Complex Architectural Design and Requirement TD* generates the most negative impact on daily software development work. The most wasted time is spent on *Understanding and/or Measuring* TD.

This study reveals that all roles are heavily affected by the interest of TD, but different roles are affected differently. The study also shows that the age of the software affects the amount of wasted time and the different activities where the time is spent on. These findings have significant implications; organizations need to be aware of how much time and resources they are spending on their interest of TD and to deliberately focus on the remediation of their TD.

REFERENCES

[1] A. Ampatzoglou, A. Ampatzoglou, A. Chatzigeorgiou, and P. Avgeriou, “The financial aspect of managing technical debt: A systematic literature review,” Information and Software Technology, vol. 64, 2015, pp. 52.

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

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

[4] T. Besker, A. Martini, and J. Bosch, "A systematic literature review and a unified model of ATD," in Euromicro Conference series on Software Engineering and Advanced Applications (SEAA), 2016.

[5] T. Besker, A. Martini, and J. Bosch, "Time to Pay Up - Technical Debt from a Software Quality Perspective," in CibSE, Buenos Aires, Argentina, 2017.

[6] Z. Codabux and B. Williams, "Managing technical debt: an industrial case study," presented at the Proceedings of the 4th International Workshop on Managing Technical Debt, San Francisco, California, 2013.

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

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

[9] R. Czaja and J. Blair, Designing surveys: a guide to decisions and procedures vol. 2. Thousand Oaks, Calif: Pine Forge Press, 2005.

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

[11] D. Falessi, M. A. Shaw, F. Shull, K. Mullen, and M. S. Keymind, "Practical considerations, challenges, and requirements of tool-support for managing technical debt," in Managing Technical Debt (MTD), 2013 4th International Workshop, 2013, pp. 16-19.

[12] C. Fernández-Sánchez, J. Díaz, J. Pérez, and J. Garbajosa, "Guiding flexibility investment in agile architecting," in Proceedings of the Annual Hawaii International Conference on System Sciences, 2014, pp. 4807-4816.

[13] Y. Guo, R. Spínola, and C. Seaman, “Exploring the costs of technical debt management – a case study,” Empirical Software Engineering, 2014/11/30, 2014, pp. 1-24.

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

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

[16] B. A. Kitchenham and S. L. Pfleeger, "Personal Opinion Surveys," ed London: Springer London, 2008, pp. 63-92.

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

[18] R. A. Krueger and M. A. Casey, Focus groups: a practical guide for applied research vol. 4. [updat]. Thousand Oaks, Calif: Sage Publications, 2009.

[19] Z. Li, P. Avgeriou, and P. Liang, “A systematic mapping study on technical debt and its management,” Journal of Systems and Software, vol. 101, 2015, pp. 193-220.

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

[21] Z. Li, P. Liang, and P. Avgeriou, "Architectural Technical Debt Identification Based on Architecture Decisions and Change Scenarios," in Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, 2015, pp. 65-74.

[22] L. M. Lix, J. C. Keselman, and H. J. Keselman, “Consequences of Assumption Violations Revisited: A Quantitative Review of Alternatives to the One-Way Analysis of Variance "F" Test,” Review of Educational Research, vol. 66, no. 4, 1996, pp. 579-619.

[23] A. Martini and J. Bosch, "The Danger of Architectural Technical Debt: Contagious Debt and Vicious Circles," in Proceedings - 12th Working IEEE/IFIP Conference on Software Architecture, WICSA 2015, 2015, pp. 1-10.

[24] A. Martini, J. Bosch, and M. Chaudron, "Architecture technical debt: Understanding causes and a qualitative model," in Proceedings - 40th Euromicro Conference Series on Software Engineering and Advanced Applications, SEAA 2014, 2014, pp. 85-92.

[25] A. Martini, J. Bosch, and M. Chaudron, “Investigating Architectural Technical Debt accumulation and refactoring over time: A multiple-case study,” Information and Software Technology, vol. 67, 2015, pp. 237-253.

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

[27] R. L. Nord, I. Ozkaya, P. Kruchten, and M. Gonzalez-Rojas, "In search of a metric for managing architectural technical debt," in Proceedings of the 2012 Joint Working Conference on Software Architecture and 6th European Conference on Software Architecture, WICSA/ECSA 2012, 2012, pp. 91-100.

[28] A. Nugroho, J. Visser, and T. Kuipers, "An empirical model of technical debt and interest," presented at the Proceedings of the 2nd Workshop on Managing Technical Debt, Waikiki, Honolulu, HI, USA, 2011.

[29] M. Ran, J. Garcia, C. Yuanfang, and N. Medvidovic, "Mapping architectural decay instances to dependency models," in Managing Technical Debt (MTD), 2013 4th International Workshop on, 2013, pp. 39-46.

[30] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Software Engineering, vol. 14, no. 2, 2009/04/01, 2009, pp. 131-164.

[31] P. Runeson and M. Höst, “Guidelines for conducting and reporting case study research in software engineering,” Empirical Software Engineering, vol. 14, no. 2, 2008//, 2008, pp. 131-164.

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

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

[34] R. K. Yin, Case study research: design and methods vol. 5. London: SAGE, 2014.