Impact of Architectural Technical Debt on Daily Software Development Work

- A Survey of Software Practitioners

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

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

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

***Abstract*— The negative consequences of Technical Debt is an area of increasing interest, and more specifically the *Architectural* aspects of it have received increased attention in the last few years. Besides the negative effects of Architectural Technical Debt on the overall software product quality in terms of hindering evolution and causing high maintenance costs, Architectural Technical Debt also has a significant negative impact on software practitioners' daily work. Although a great deal of theoretical work on Architectural Technical Debt has been undertaken, there is a lack of empirical studies that examine the negative effects of *Architectural* Technical Debt during the software development lifecycle. The aim of this study is to investigate how practitioners perceive and estimate the impact of Architectural Technical Debt during the software development process. This paper reports the results of an online web survey providing quantitative data from 258 participants. The contribution of this paper is threefold: First, it shows that practitioners experience that the *Architectural* type of Technical Debt has the highest negative impact on daily software development work. Secondly, we provide evidence that does not support the commonly held belief that Architectural Technical Debt increases with the age of the software. Thirdly, we show that despite different responsibilities and working tasks of software professionals, Architectural Technical Debt negatively affects *all* roles without any significant difference between the roles.**

Keywords-component; Architectural Technical Debt, Wasted time, Software Age, Software Roles, Software Development, Empirical Study, Survey, Quantitative data

I. INTRODUCTION

Technical Debt (TD) is recognized as a critical issue in today's software development industry [1]. Since this phenomenon has a negative effect on software companies, it is important to assess and estimate the consequences of TD, both in terms of the wasted time it causes and its negative effects on daily software development work.

Ward Cunningham [2] introduced the financial metaphor of TD to describe to nontechnical product stakeholders about the need to identify the potential long-term and far-reaching negative effects of the immature code that is implemented during the software lifecycle.

A more recent explanation was provided by Avgeriou et al. [3] who describe 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.*"

The reason for taking on TD can be defined as an unintentional consequence resulting from the accumulation of non-optimal decisions over time or, as in Cunningham's definition, intentionally in order to get new functionality running quickly [4].

Interest is the negative effects of the extra effort that has to be paid due to the accumulated amount of TD in the software, such as executing manual processes that could potentially be automated or expending excessive effort on modifying unnecessarily complex code, performance problems due to lower resource usage by inefficient code, and similar costs [1],[4]. The interest can also be described as "*The additional effort that is needed to be spent on maintaining the software because of its decayed design-time quality*" [5].

Today, it has been well established from various studies that TD has a negative impact on software development organizations by impeding evolution and causing high maintenance costs due to internal software quality issues [6], [7], [8], [9], [10]. Left unmanaged, TD can result in significant cost overruns, severe quality issues, a limited ability to add new features [11] and even result in reaching a crisis point when a huge, costly refactoring or an entire replacement of the system needs to be undertaken [12].

During development of large-scale software systems, the software architecture plays a significant important role [6] and consequently a vital part of the overall TD relates to sub-optimal architectural decisions and is regarded as *Architectural* Technical Debt (ATD) [13].

ATD is primarily incurred by architectural compromises with the consequence of immature architectural artifacts such as violations of best practices [14], consistency and integrity constraints of the architectures, or implementation of immature architectural techniques. These concerns chiefly result from the compromise of performance efficiency, reliability, maintainability, reusability, and the ability to add new features [15].

Besides the negative effects ATD has on the overall software product quality, ATD also has a significant negative impact on software practitioners' daily work. It is, therefore, important to understand *when*, and to *whom,* and in *what way* (in terms of generating wasted time) ATD has a negative impact during the software development process. In this study, the negative effects of ATD from a software lifecycle perspective are examined by conducting a web survey of 258 industry practitioners. This paper has four key goals:

Firstly, there are no studies quantifying the interest in terms of how much (observed, measured or estimated) time is wasted due to *Architectural* TD. We aim to understand the level of negative effects ATD has on the daily software development work and compare this level with negative effects due to other types of TD.

Secondly, we aim to understand if the level of the negative effects due to ATD correlates with the estimated wasted development time during the software lifecycle.

Thirdly, the negative effect due to ATD is commonly believed to have an increasingly negative impact with respect to the age of the software. This study aims to assess the extent to which the level of negative effects due to ATD, differs in relation to the age of the system.

Finally, during the software lifecycle, several different professional roles participate, and could subsequently be affected differently by ATD. To the best of our knowledge, no previous studies have examined how different software professional roles perceive the negative effects specifically generated by ATD. Therefore, this paper analyses the variation in the negative impact of ATD on different software roles on their daily software development work.

With regards to these four goals, we aim to answer the following research questions (RQ):

RQ1. *Does ATD generate a negative impact on daily software development work?*
RQ2. *Does ATD negatively affect the amount of wasted development time?*
RQ3. *Does the Age of the software system affect the negative effects due to ATD?*
RQ4. *In what way are different software Roles affected by ATD?*

The research questions will be answered using estimated 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 companies.

This study makes an original contribution to research, with respect to the existing body of knowledge relating to ATD in the following important areas:

1) We provide an empirically based study on how practitioners estimate and experience ATD within the software industry, based on empirical quantitative data.

2) We present results that confirm that ATD has a high negative impact on the practitioners' daily software development work.

3) This estimated wasted time by the practitioners does not correlate with the level of negative impact generated by ATD, on daily software development work.

4) This study shows that the level of negative impact due to ATD is introduced early, and thereafter remains during the whole software lifecycle. Based on evidence from our survey, this study does not support the currently held belief that the negative effects due to ATD increase with respect to the age of the system.

5) This study provides new insights into ATD research by showing that ATD has an extensive negative effect on *all* software professional roles.

6) This study demonstrates to both practitioners and academics the importance of paying more attention and effort to early remediate ATD during the software lifecycle, in order to decrease the level of negative impact due to ATD on daily software development work.

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

II. RELATED WORK

Research addressing the negative effects and impacts of TD in general is relatively well represented within academia [16], but there is currently no academic research quantifying how much software development time is estimated to be *wasted* due to *Architectural* TD and if such wasted time varies in relation to the level of the experienced and estimated negative effects generated by ATD. A previous study by Li et al. [17] states that ATD concerns different types of roles in different ways; however, this current study fills a gap in the literature by addressing to what extent different professional roles perceive the negative effects of ATD and exploring how these negative effects vary during the software lifecycle, in terms of the system age.

A. The importance of addressing ATD

ATD is an important component that needs to be addressed in the architecting process [17], in order to make sure that the advantages of sub-optimal solutions do not lead to large future payments of interest [18], [19], [20].

The importance of ATD has been highlighted in several studies with statements such as, "*the most encountered instances of technical debt are caused by architectural inadequacies*" by [21], "*we found that architectural decisions are the most important source of technical debt*"

by [22], and "*because architecture has such leverage within the overall development lifecycle, strategic management of architectural debt is of primary importance*" [23]. In our previous paper [12], we pointed out the risks when ATD grows until the result of negative effect causes adding new business value so slowly that it becomes necessary to conduct extensive refactoring or even rebuilding the complete software from scratch.

B. Surveys-based studies on TD

By reviewing other survey-based studies focusing on TD, we, in particular, found four studies which bear a resemblance to this study. First, Ernst et al. [22] conducted a survey within three large organizations, with 536 respondents (primarily software engineers and architects). This research included follow-up interviews with seven software engineers. Similar to this study, one of their main research areas focused on how much of the TD is architectural in nature. Their result shows that "bad architecture choices" are the greatest source of TD. The study also determines that the degree of architectural drift is related to system age. They found a weak association between the age of the system and the perceived importance of architectural issues where 89% of the respondents with software age ≥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 systems (< 3 years). Compared to this study, our study has a higher granularity of the age intervals and therefore provides a more detailed description from a lifecycle perspective. In contrast to that study, our study also includes the estimated wasted time due to ATD.

Secondly, Holvitie, Leppanen and Hyrynsalmi [21] conducted a web survey of Finnish software practitioners to determine their level of TD knowledge, how TD manifests in their projects, and which of the applied components of agile software development are sensitive to TD. Consequently, they found that the most frequently indicated causes of TD were inadequate architecture and inadequate documentation. This study was conducted in Finland and included 54 applicable responses. However, in these studies, we cannot find a quantification of the interest, and furthermore, there are no examinations on the consequences ATD specifically has for different software roles.

This paper is, to some extent, also related to our previous papers [24],[15], where we study and compare several different TD types and address how TD negatively affects different software quality attributes. However, even if the data are collected using the same survey, this study focuses on the *Architectural* aspects of TD, and does not include any effects on the software quality attributes, and does not focus on other types of TD. By only focusing specifically on ATD, this means that we can provide a more in-depth analysis of the architecturally related issues of TD and provide more detailed statistical analysis of the data.

C. Research on TD interest variations

Kazman et al. [25] have published a case study 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 report the cost of interest, but instead targets 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. By comparison, this study has a focus on quantifying the interest in terms of wasted time instead of on locating the roots of the architectural TD and the expected benefit from the refactoring.

Falessi et al. [26] argue that the interest of TD can change over time, is non-linear and has a probability of growing exponentially rather than linearly. By examining source code, Nugroho et al. [27] state that the interest grows differently on a 10-year horizon, depending on a star rating system for the software.

Xiao et al. [28] proposed an approach to identifying and quantifying ATD, focusing on modeling the growth and on the quantification of the maintenance cost. In this study, the interest is calculated using approximations based on the number of lines of code modified and committed to fix bugs. This work has the limitation of only including the cost of maintenance as ATD and primarily focuses on bug fixes.

Although some research has been conducted on addressing the growth of the interest, there is a lack of empirical investigations quantifying the interest from an architectural perspective in terms of wasted time.

III. METHODOLOGY

The following sections describe the methods used for conducting the survey, the data collection, and the data analysis.

A. Design of the Survey

The web survey was designed and hosted by the online survey service *Survey Monkey*. The questions were a combination of optional and mandatory. To avoid bias in the survey, the questions were developed as neutrally as possible, in such a way that one question did not influence the response to the next, and clear, unbiased instruction was provided when needed [29].

The survey was divided into two different sections in order to give the respondents a context and understanding for each set of questions. The first draft of the survey was tested by four industrial practitioners (developer, manager, project owner and software architect) and by two Ph.D. students in order to evaluate the understanding of the questions and the usage of common terms and expressions [30]. During this evaluation, we also monitored the time that was needed. 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. All invited respondents were informed about the survey with an invitation text describing the motives and goals of the study.

B. Data collection of survey data

The invitation to the survey was emailed directly to seven companies/partners within our networks (located in Scandinavia, with an extensive range of software development), and invitations were also distributed 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%), the decision was made to reject the unfinished questionnaires, according to guidelines by [29].

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 relatively high, with 58% having a Master's degree and 25% having a Bachelor's degree. The survey included respondents having different professional roles, where 49% were Developers/Programmers/Software Engineers (abbreviated to Developers in this paper), while 25% were Software Architects. Approximately 78% of the Software architects and 59% of the Developers/Programmers/Software Engineers had more than 10 years of experience. The most common size of the software development team included 6-10 members (36%), and the majority (32%) systems were on average 5-10 years old from the initial design. Nevertheless, a significant number (35%) of the respondents' software was more than 10 years old. As indicated in Table I, the majority of the software systems were embedded systems (49%) and real-time systems (42%). However, for this specific question, the respondents could select more than one option.

TABLE I. CHARACTERISTICS OF THE SAMPLE SURVEY

Individual data		Company data	
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%
		> 40 members	18,20%
Education			
Master's 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%
		Modeling and/or simul.	15.12%
Roles		Data analysis system	14.73%
Developer/Program/-Software Engineer **	49,20%	Web 2.0 / SaaS system	8.53%
Software Architect	24,80%	Other	8.53%
Manager	6,20%		
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%
		10-20 years	28,3%
Gender		>20 years	6,6%
Male	89,90%		
Female	8,50%		
Other/no share	1,60%		

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

The second part of the survey focused on questions based on the research questions presented in Section I. In this part of the survey, the following survey questions (SQ) were asked:

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 time spent fixing and managing the issues, just if they hinder you when you add/change/understand the system).

For the first question SQ1, the listed categories and sub-categories (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, Uneasy/Tensed social interactions between different stakeholders) provided by [31] were used to distinguish between different TD types. All these TD types were referred to as "Challenges", in order to provide the respondents with a more detailed clarification of the TD type. This also mitigated the risk of mis-interpretations and helped to make sure that the respondents were considering the correct type of TD issue.

In order to ensure a better construct validity of the survey, these same types of TD were referred to in survey question SQ2. In SQ2, the estimation of the interest as wasted time was specified 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 TD issues.

1) Data analysis of survey data

The data from the survey were analyzed using a quantitative method, by interpreting the numbers obtained from the answers to the survey. All statistical 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.

The data were 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 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.

IV. RESULTS AND ANALYSIS

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

A. *Does ATD generate a negative impact on daily software development work?*

In order to understand the level of negative impact ATD has on the practitioners' daily software development work, this TD type was compared with several other TD types. The survey respondents were asked to rank (score 1 = lowest negative impact, score 11 = highest impact) a randomly ordered list of different TD types they consider to have the most negative impact on their daily software development work. Each of these challenges corresponds to different TD types which reflected the TD types that emerged from [31]. For example, if one of the TD types was ranked as generating the *most* negative effect by one respondent, it was given a score of 11, and if second 10 and so on, and the score 1 was given if the negative impact was ranked as generating the *least* negative effect.

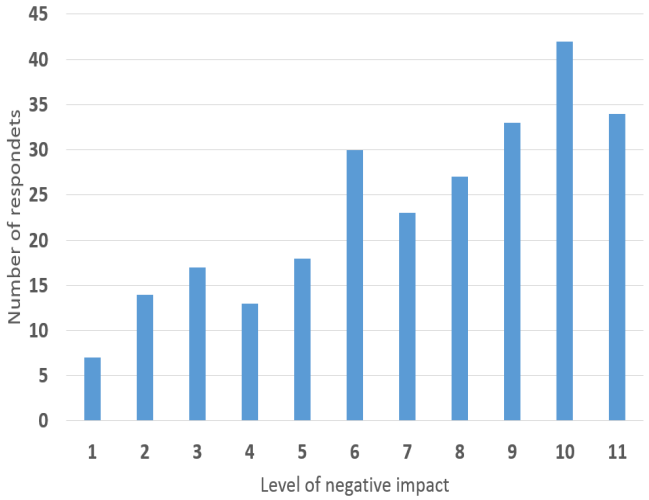

Figure 1. Distribution of the negative effects due to ATD

The results of the level of negative effects due to ATD in terms of Complex Architectural Design (CAD) are presented in Fig 1. From the data in this figure, it is apparent that the number of respondents is increasing in relation to the perceived negative impact of ATD. A total of 109 (42%) of all the respondents experience the three highest levels of negative effect (score 9, 10 and 11) due to ATD. Only seven (2.7%) respondents perceive that ATD has the least negative impact, while 34 (13.2%) respondents estimate ATD has the most negative impact on their daily work.

In our previous study [24], we studied how several different types of TD have an negative effect on the practitioners' daily software development work. To understand the impact of ATD, we have, in this study, compared the negative effects due to ATD with data from the previous study.

Table 2 presents the summary statistics for each of the listed challenges in the survey, based on the results of the other TD types presented in our previous study [24]. The mean value for each TD type, is the average of the weights associated with the ranking reported by the respondents in the survey. Given these weights, a mean value was calculated for each TD type: this way, we could see which challenges were ranked as having the most and least negative impact on the daily software development work.

Table 2 reveals that CAD (mean rank 7.27 with a standard deviation of 2.9) generates the most negative impact on daily software development work, both when studying the mean and the median values. The level of negative impact due to a complex architectural design is closed followed by Requirement Technical Debt (mean rank 7.23).

TABLE II. MEAN AND MEDIAN OF THE RANKINGS FOR TD TYPES

Challenge	***Mean***	***Median***
Complex Architectural Design	7,27	8
Requirement TD	7,23	8
TestingTD	6,80	7
Code TD	6,36	7
Documentation TD	5,98	5,5
Many patterns and policies	5,76	5
Dependency Violations	5,71	6
Infrastructure TD	5,59	5
Lack of reusability	5,38	5
Dependencies to externals	5,19	4
Social TD	4,73	4

B. *Does ATD negatively affect the amount of estimated wasted development time? (RQ2).*

This research question focuses on the interest in terms of how much of the overall software development time is wasted due to paying the interest of ATD. In our previous paper [24], we reported that the respondents estimate that ***36%*** of all software development time is wasted because of TD in general, with the standard deviation of 17.8%.

In this specific research question, we seek to confirm the commonly held belief in a correlation between a higher level of negative effect and a higher amount of wasted development time.

To evaluate if there is such a significant correlation between the estimated overall wasted development time and the experienced level of negative impact due to ATD, we have correlated these two datasets.

When testing the belief that the estimated wasted time increases in relation to the level of negative impact generated by a complex architectural design, in terms of ATD, we calculated the average estimated wasted time within each of the levels of the negative impact (the same ranking scale as in Section IV.A, were used where level 1 = lowest negative impact and level 11 = highest negative impact due to ATD).

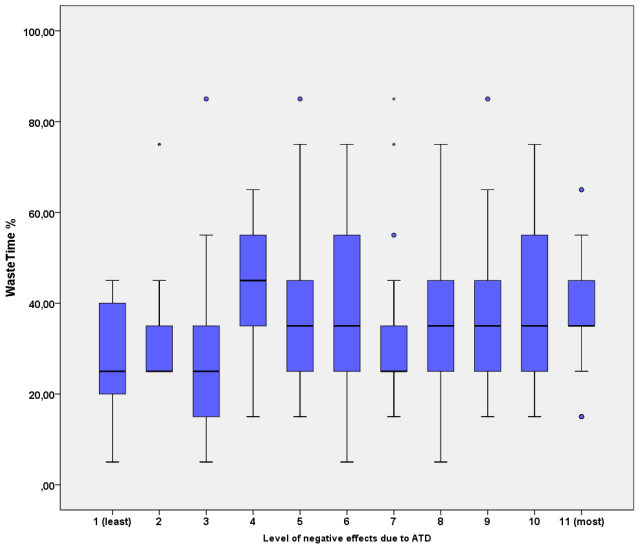

Figure 2. Estimated wasted software development time for each level of negative effects due to ATD

Fig. 2 represents a boxplot for each level of the negative effect due to ATD. In this figure, the estimated wasted time is plotted and compared using the different levels of negative impacts on the daily software development work due to ATD.

This boxplot represents the minimum and maximum range values, the upper and lower quartiles, and the median for each level. From the boxplot, we can see that the estimated wasted time is relatively consistent within the different levels of negative impact. The maximum estimated wasted time (41%) was found within level 4 of the negative impact and the minimum wasted time (30%) was found in the first and the third level of negative effects due to ATD.

In Fig. 3, the differences of distributions of the estimated wasted time in relation to the mean value of the negative effect generated by ATD are graphically visualized. Respondents who estimate they are wasting less than 10% of their working time due to TD, estimate the lowest negative effect due to ATD (mean value 4.71) and respondents estimate they are wasting the most wasted time (80-90%) estimate the second lowest negative effect (mean value 6.0).

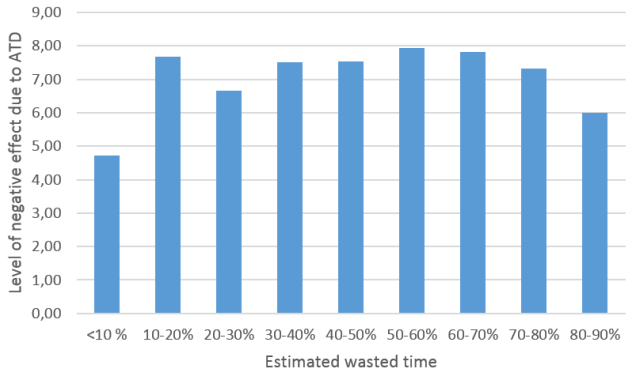

Figure 3. Mean value of the negative effect due to ATD in relation to the estimated wasted time

The analysis determined by one-way ANOVA showed that the estimated wasted time was not statistically significantly different between the levels of impact.

Furthermore, to evaluate if there is a linear correlation between the level of the negative impact caused by ATD and the wasted time, we used the Pearson correlation method. These two data series (we used the average wasted time within each interval), returned p-value 0,084, which showed no support for a linear correlation between the estimated wasted time and level of negative impact due to ATD.

Overall, this analysis did not confirm a statistically significant correlation between the estimated wasted time and the level of experienced negative impact ATD has on the respondents' daily software development work.

This result indicates that the respondents who stated that ATD generates the most negative effects (rank 11) on their daily work, did not waste significantly more or less amount of time compared to the respondents stating that ATD generates less negative effects.

In summary, the perceived negative effects due to ATD did not affect the respondents' estimated wasted time and, based on evidence from our survey, this study does not support the currently held belief that the negative effects due to ATD significantly correlates or linearly increases with respect to the age of the system.

C. *Does the Age of the software system affect the level of negative effects due to ATD? (RQ3)*

There is a commonly held belief that the negative effects of a complex architectural design, in terms of ATD, increase with the age of the software, commonly referred to as software aging [32].

In order to assess whether the negative effects of ATD varies during the software lifecycle, we used a statistical cross-tabulated analysis showing the negative effects caused by ATD (using the same ranking scale as in Section IV.A) on the daily software work in relation to different software age intervals of < 2 years, 2-5 years, 5-10 years, 10-20 years, and > 20 years.

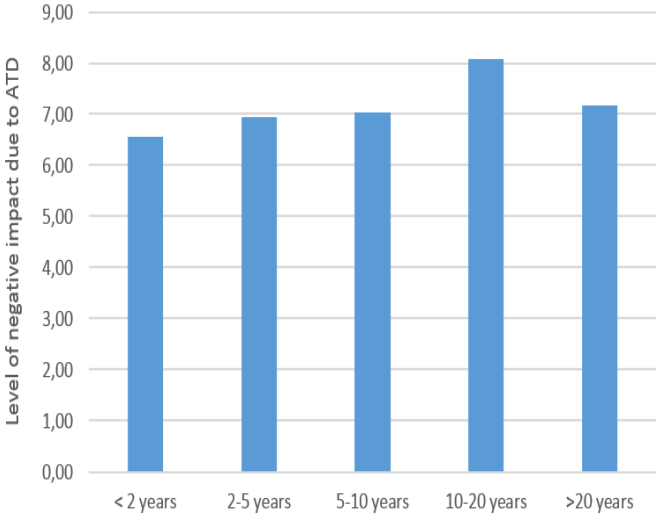

Figure 4. Distribution of negative effect (mean values) for each software age interval

To calculate the mean value for the ages of the software, the average of the reported age interval stated by the respondents in the survey was used. For example, if the age was reported within the interval of < 2 years, the average age was set to 1 year, and for the interval 5-10 years, the average age of 7.5 years was used. For systems reported > 20 years the age of 25 was used.

The first set of analyses examined the negative effects due to ATD for each software age interval and, as shown in Fig. 4, the mean values of the negative effect somewhat vary over the different age intervals. What is interesting about the data in this figure is that, for software which is less than two years, the negative impact of ATD has already a relatively high negative impact on the respondents' daily software work, with a mean value of 6.56. This result implies that ATD is introduced quite early during the software lifecycle and is also generating extensive negative effects for systems with ages less than two years. After the first two years, the negative effects have a growing trend up to the peak (mean value 8.08) in the age interval of systems with ages between 10 and 20 years. For systems older than 20 years, the negative impact somewhat decreases (mean value 7.18).

To further evaluate if there is a linear correlation between the system age and level of negative impact, we used the Pearson correlation method. Despite the visual impression in Fig. 4 somewhat indicating a small increase in the software age of the system, our statistical analysis of the two data series (we used the average age within each year interval) did not return any indication of a significant linear correlation between the system age and the level of negative effects due to ATD (r(256)=,127, *p*=0,041). The boxplot in Fig. 5 captures the difference in software age in relation to each level of negative impact due to ATD. Looking at the comparison of median and percentiles among the different levels of negative impact, we cannot see a significant difference with regards to the age distribution, besides for level 10 (median 15 years) and level 5 (median 5.5 years). The mean values of the system age vary between a minimum of 6.1 years (for level 5) and a maximum of 12.1 years (for level 10) among all the different levels. To evaluate if the software age is significantly different among the different intervals, a one-way ANOVA test was used. This test revealed that the age of the system is significantly different among the different levels of impact (F(10,247)=1,928, p=0.042). Furthermore, the Pearson chi-square test of dependence showed that the two factors (system age and level of negative impact) are not dependent, which means that there is no significant relationship between the age of the system and the experienced level of negative effects due to ATD. Hence, we did not find any statistically verifiable evidence showing a correlation between the age of the software and the level of negative effects generated due to ATD on the respondents' daily work. The result shows that, for young software systems with an age of less than two years, the negative impact generated by ATD is also quite extensive. Based on evidence from our survey, this study does not support the currently held belief that the negative effects due to ATD, significant correlates or linearly increases with respect to the age of the system.

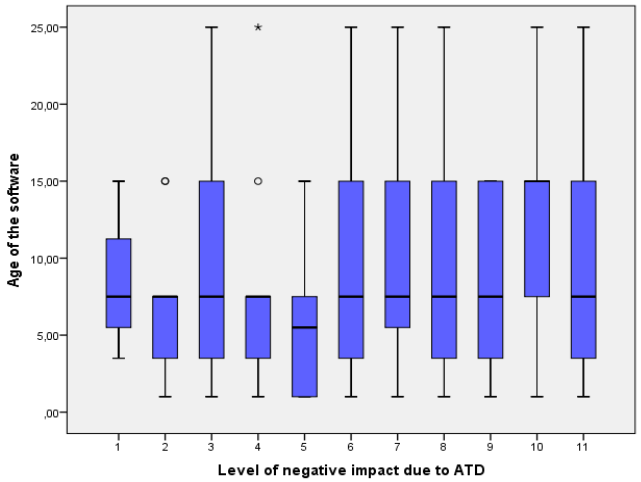

Figure 5. Age of the software for each level of negative effects due to ATD

D. *ATD impacts on different Roles (RQ4)*

In our previous study [24], we found that, among all different TD types, ATD has the greatest negative impact on the daily software development work for all the different roles. In this section, we explore if the different professional software roles (Developer, Software Architect, Manager, Project Manager, Product Manager, Expert, and Tester) experience the negative effects generated by ATD, on their daily software development work differently.

This research question uses the same ranking scale as in Section IV.A, where level 1 represents the lowest negative impact and level 11 represents the highest negative impact generated by ATD.

By examining the mean values in Fig. 6, it is evident that Project Managers (mean value 8.31) and Testers (mean value 8.11), estimate the highest negative impact due to ATD whereas Experts (mean value 6.46) and Software Architects estimate the lowest value of the negative impact of ATD (mean value 6.67).

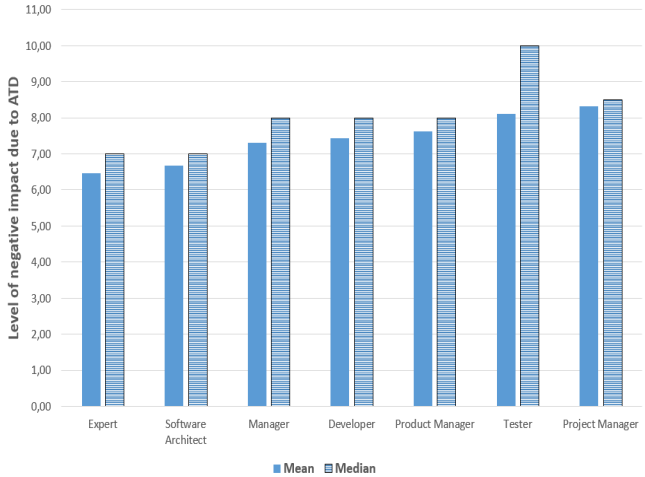

Figure 6. Different roles' level of negative effects due to ATD

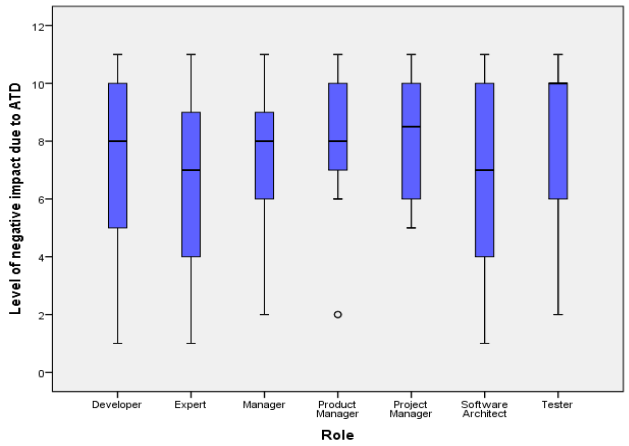

Figure 7. Distribution of different roles' level of negative effects due to ATD

To graphically visualize the differences of distributions for each negative impact level, Fig. 7 presents a boxplot for each of the roles. From this boxplot, we can see the variation and distribution of how the roles estimate the negative impact due to ATD.

When explicitly tested if the roles experience the negative effects due to ATD differently, a one-way ANOVA test showed, somewhat surprisingly, that there were no significant differences ($F(6,251)=1,16, p=0.310$) between how the roles are negatively affected by ATD.

One unanticipated finding was that five of the Software Architects (8%) stated that ATD generates the least (level 1) negative impact on their daily work. Meanwhile, 19% of the Project Managers perceive that ATD has the highest level of impact on their daily software development work.

The result of this research question, indicates that *all* different software professional roles are extensively negatively affected by ATD.

V. Discussion and Limitations

This section presents and discusses the findings and the implications of the research questions presented in Section I.

A. The negative impact on daily software development work due to ATD (RQ1)

Several previous studies have described that ATD has a severe negative impact on software development in general, but there is, to the best of our knowledge, no research specifically focusing on quantifying the level of negative impact ATD has on practitioners' daily software work.

The first research question (RQ1) aims to address the level of the negative impact ATD has on the respondents' daily software development work. The result shows that ATD is ranked as having a high negative impact on the practitioners' daily software development work. Moreover, compared to other types of TD [24], this survey reveals that *Complex Architectural Design,* closely followed by *Requirement TD,* generates the most negative impact on daily software development work. This finding broadly reflects the findings from other studies in this area by describing that TD instances linked to inadequacies in the software architecture are major sources of TD [21], [22], [33], [34].

B. Does ATD negatively affect the amount of wasted development time? (RQ2).

The second research question (RQ2) focuses on how much of the overall software development time is estimated to be wasted because of ATD and also to evaluate the commonly held belief of a correlation between this estimated wasted development time and the experienced level of negative impact generated by ATD.

A common belief exists that there is a correlation between the estimated wasted software development time and the negative effects due to ATD, but this study provides empirical evidence which does not support such belief. The level of negative effects generated by ATD thus does not have a significant correlation between the amount of estimated wasted software development time. In contrast, the estimated wasted time did not differ significantly among the different levels of negative effects generated by ATD. However, the estimated wasted time varies in relation to the level of negative effects due to ATD, but the distribution of the wasted time did not have a significant increase (or decrease) linear correlation with respect to the different levels of negative impacts.

Contrary to popular belief, how negatively respondents estimate the impact due to ATD proves to be a poor indicator of wasted software development time.

Therefore, studies asking software professionals how often they encounter ATD, how they perceive the importance of ATD, or how they experience the negative impact due to ATD, should be carefully examined. This is because our research shows that there is no correlation between the level of negative effects due to ATD and the amount of the estimated wasted time.

Further research should be undertaken to investigate if other types of TD (besides ATD) have a significant correlation to the estimated wasted time to better understand the negative impact different TD types have on the wasted software development time.

C. Does the Age of the software system affect the negative effects due to ATD?(RQ3)

The third research question (RQ3) in this study sought to determine if there is a correlation between the level of negative impact generated by ATD and the age of the software.

It is a commonly held belief that the negative effects generated by ATD increase over time within the software lifecycle. However, the findings of this study do not support such a belief. The negative effects due to ATD vary in relation to the system age (but not linear), and were observed as having the highest negative impact on software with an age interval of 10-20 years and the lowest value for the age interval of < 2 years.

What is interesting about the result is that for young software with an age of less than two years, the negative impact generated by ATD is also quite extensive.

A possible explanation for these results may be that ATD is introduced early in the software, and then persists throughout the whole software lifecycle. This result implies that investment in refactoring must be continuous from the conception of the system in order to keep the negative effect generated by ATD at a future low level.

D. ATD impact on different Roles (RQ4)

The fourth research question (RQ4) explores the negative effects ATD has on different types of professional software roles. Both technically oriented professionals such as developers, testers, and experts and also professionals working within a management area of the software, such as software architects, product and project managers, participate in the software development process during the software lifecycle. All these roles have different responsibilities and tasks and could subsequently potentially be differently affected by ATD. The statistical analysis could not confirm the negative impact generated by ATD among the roles as significantly different. This study confirms that ATD has an extensive negative effect on *all* software professional roles within the companies.

5) Verifiability and Limitations

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/230742#.WG34rP7rupp. The reported findings are subject to at least two limitations. First, the majority of the data from the invited companies for the survey and the interviewed companies were gathered in Scandinavia. Thus, the results might be different in other geographical and cultural areas. As a result, further work is needed to replicate the results in other geographical areas and software development cultures.

Second, the qualitative data derived from the survey are not based on measured or observed data but rather on estimations made by the respondents. Even if there was a high degree of agreement across the respondents' estimations, this might create bias for the results in terms of their credibility and correctness. Assessment of TD is, however, not an exact science; it all depends on how you calculate it [35]. Therefore, as a future study, we plan to investigate this area further, to also include physical measurements and observations, in order to create a stronger reliability of the data.

VI. Threats to Validity and Verifiability

The result of this research may be affected by some threats to validity, such as construct validity, internal validity, external validity, and reliability.

Construct validity reflects to what extent the operational measures that are studied represent what the researchers have in mind, and what is investigated according to the research questions [36]. 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 Challenge.

This study could potentially suffer from *internal validity* when the causal relationships were examined, as it affects our ability to accurately explain the phenomena that we observed [37].

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 primarily investigated companies from the Scandinavian area. To mitigate this validity issue, we attempted to enlarge the respondents' sample by inviting additional participants globally via LinkedIn. Without replicating this study to other countries or regions, 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 observer triangulation, which means that all authors participated in the analysis.

VII. Conclusion

Architectural Technical Debt is evidently detrimental to software companies, and it is important to assess and estimate its negative consequences on daily software development work. The main goal of this study was to determine how practitioners within the software industry perceive and estimate the negative effects due to ATD in terms of the level of negative effects.

To the best of our knowledge, this is the first study that empirically surveys the estimated detrimental impact of ATD in terms of estimated wasted time and effort. This study is based on a survey with 258 respondents.

The study has shown that ATD has a high negative impact on the practitioners daily software development work. The research has also shown that the amount of wasted time does not have a statistically significant linear correlation with how the practitioners experience the level of negative effects due to ATD. Practitioners waste time statistically independently of how they experience the negative impact of ATD.

Furthermore, this study reveals that ATD has an extensive negative effect on *all* software professional roles and, additionally, our study provides evidence which does not appear to support the commonly held belief that the ATD increases with the age of the software, as this study did not find any statistically verifiable relationship between the age of the software and the level of negative effects generated by ATD. The result shows that ATD is introduced early and persists during the whole software lifecycle. These findings have significant implications, advocating for the important awareness of how much valuable time and effort is wasted due to ATD.

These findings provide strong empirical confirmation that software companies need to invest in continuous refactoring from the conception of the system in order to keep the negative effect generated by ATD at a future low level.

REFERENCES

[1] 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.

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] A. Martini, J. Bosch, and M. Chaudron, "Architecture Technical Debt: Understanding Causes and a Qualitative Model," in Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on, 2014, pp. 85-92.

[15] T. Besker, A. Martini, and J. Bosch, "Time to Pay Up - Technical Debt from a Software Quality Perspective," in Accepted at the 20th Ibero-American Conference on Software Engineering, CibSE, 2017.

[16] 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.

[17] Z. Li, P. Liang, and P. Avgeriou, "Chapter 5 - Architecture viewpoints for documenting architectural technical debt," in Software Quality Assurance, I. M. S. A. G. Tekinerdogan, Ed., ed Boston: Morgan Kaufmann, 2016, pp. 85-132.

[18] U. Eliasson, A. Martini, R. Kaufmann, and S. Odeh, "Identifying and visualizing Architectural Debt and its efficiency interest in the automotive domain: A case study," in Managing Technical Debt (MTD), 2015 IEEE 7th International Workshop on, 2015, pp. 33-40.

[19] A. Martini and J. Bosch, "Towards Prioritizing Architecture Technical Debt: Information Needs of Architects and Product Owners," in Software Engineering and Advanced Applications (SEAA), 2015 41st Euromicro Conference on, 2015, pp. 422-429.

[20] N. Brown, Y. Cai, Y. Guo, R. Kazman, M. Kim, P. Kruchten, et al., "Managing technical debt in software-reliant systems," presented at the Proceedings of the FSE/SDP workshop on Future of software engineering research, Santa Fe, New Mexico, USA, 2010.

[21] 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.

[22] 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.

[23] P. Kruchten, "Strategic management of technical debt: Tutorial synopsis," in Proceedings - International Conference on Quality Software, 2012, pp. 282-284.

[24] T. Besker, A. Martini, and J. Bosch, "The pricey Bill of Technical Debt - When and by whom will it be paid? ," in submission, 2017.

[25] 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.

[26] 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.

[27] 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.

[28] L. Xiao, Y. Cai, R. Kazman, R. Mo, and Q. Feng, "Identifying and quantifying architectural debt," in Proceedings - International Conference on Software Engineering, 2016, pp. 488-498.

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

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

[31] 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.

[32] D. L. Parnas, "Software aging," in Software Engineering, 1994. Proceedings. ICSE-16., 16th International Conference on, 1994, pp. 279-287.

[33] 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.

[34] 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.

[35] A. Szykarski. (2012). Ted Theodorpoulos on Managing Technical Debt Successfully, Available from: http://www.ontechnicaldebt.com/blog/ted-theodoropoulos-on-managing-technical-debt-successfully/.

[36] 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.

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