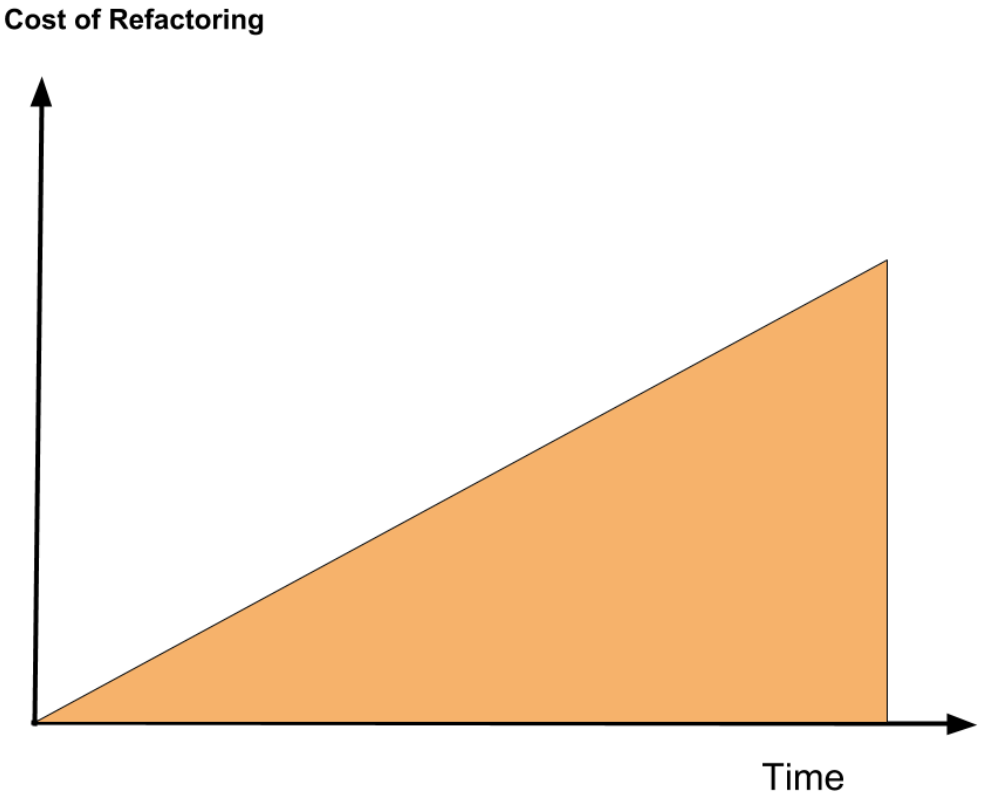
CHALMERS UNIVERSITY OF TECHNOLOGY | UNIVERSITY OF GOTHENBURG



Estimating Architectural Technical Debt

A Design Research

Master's thesis in Software Engineering

GUSTAV DAHL

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY & UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2017

Master's thesis 2017:NN

Estimating Architectural Technical Debt

A Design Research

GUSTAV DAHL

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology & University of Gothenburg
Gothenburg, Sweden 2017

Estimating Architectural Technical Debt
A Design Research
GUSTAV DAHL

Supervisor: ANTONIO MARTINI and JAN BOSCH, Department of of Computer Science and Engineering
Examiner: ERIC KNAUSS, Department of Computer Science and Engineering

Master's Thesis 2017:NN
Department of Computer Science and Engineering
Division of Software Engineering
Chalmers University of Technology & University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Graph showing the cost of refactoring will change over time.

Typeset in LaTeX
Gothenburg, Sweden 2017

Estimating Architectural Technical Debt
A Design Research
GUSTAV DAHL
Department of Computer Science and Engineering
Chalmers University of Technology & University of Gothenburg

Abstract

Technical debt(TD) and the sub-category architectural technical debt (ATD) are two software related buzzwords frequently used in both academia and in the software industry. The purpose of these terms is to make it easier to understand that a software decision might lead to an expected or unexpected consequence that could have an impact in the long-run. Hence, the TD level in a project needs to be under control. However, the common approach towards refactoring of a TD is to handle it when it is too late and a crisis has emerged due to its presence. In order to solve this and make the stakeholders able to determine when a TD should be refactored a tool has been developed. This tool incorporates the newly developed AnaConDebt model, which is an ATD refactoring decision model. The outcome from building and evaluating this proof-of-concept is that there is potential for such a tool but it is not yet there. The underlying model needs to be further developed incorporating more info used by the industry.

Keywords: Technical debt, Architectural Technical debt, Software Architecture, Refactoring, Refactoring decision

Acknowledgements

I would first and foremost like to thank my supervisors at Chalmers University of Technology, Antonio Martini and Jan Bosch. They helped me from start to end by guiding me in how to conducted the research in a scientific manner and then summarizing it into the report you are currently reading. I would also like to thank my supervisor Ali at the case company for setting a high bar for this thesis and expecting nothing but excellence. Other people who deserves praise for their contributions are the people at the case company, Christofer, Martin, Victor, Peter, who has been helping me through out the thesis and made me feel very welcome.

Gustav Dahl, Gothenburg, June 2017

Contents

List of Figures

List of Tables

1

Introduction

In 1992, Ward Cunningham expressed the term technical debt(TD) for the first time. He used it as a metaphor to explain the phenomena regarding releasing a software project for the first time to the public. To be more precise, he aimed to explain how the software release would affect the quality of the source code and as a result affect the future development of the project. The comparison that the metaphor is trying to make is that releasing code for the first time is like going into debt. The released software might satisfy the customers needs at this point in time, but sub-optimal code decisions done in order to meet the release date will lead to an extra cost for the development in the future. [1]. In other words, meeting short-term goals often leads to going into debt, and the interest is the costs of repairing it to meet long-term goals. [3] TD is a flexible metaphor and a various amount of sub-categories have been expressed in order to address different kinds of software related debts. A few examples of these kind of sub-categories are Requirements debt, Document debt and Architectural technical debt. [2] The scope for this thesis is Architectural technical debt(ATD) and can be described as contraventions against the intended and pre-defined software architecture. [6]

Figure 1.1 shows the current technical debt landscape as suggested by Phillipe Krutchen, Robert L.Nord and Ipek Ozakaya. On the right side of the blue box are code related attributes which shows themselves as quality issues in the product. These issues can be found using static code analyzation tools. A few examples of such code analyzers are NDepend or SonarQube. [18][19] On the other hand, to the left in the red box are the architecture and structural issues with the evolvability of the product. These types of issues are very common TDs in a software project and it is therefore important to take care of them when developing larger projects that will be supported after development has finished. However, these issues are unfortunately not found using the common code analyzation tools. [2][11]

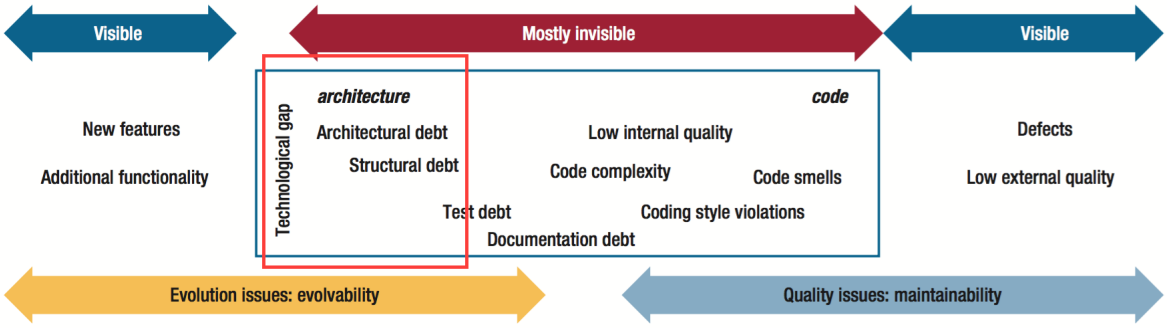


Figure 1.1: The technical debt landscape. The image is borrowed from Phillippe Krutchen, Robert L.Nord and Ipek Ozkaya [2]

In order to measure quality of the source code (the right side of figure 1.1) various amounts of Software quality metrics and software quality models have been defined. A software quality metric measures something very specific regarding the software. A few examples of these metrics could be lines of code or coupling. [20] However, it is important to be aware of that only using one metric for measuring software quality does not provide a good picture. Several metrics should instead be used together in order to triangulate the result. A software quality model is a definition of Software characteristics that should be measured and together in total create a credible software quality framework. Because of its complexity there are fewer models developed for the architecture related software quality (the left side of figure 1.1)but one of the more commonly used example of such method is the Architecture Tradeoff Analysis Method(ATAM). [21]

Even though the models developed are effective, none of them provide any reasoning for whether it is a good decision to refactor from a cost-wise perspective. This is something that the software industry has acknowledged as a major issue related to technical debt. They can get an estimated cost of the technical debt, but how do they know if it is actually worth spending time on fixing that particular TD? Will the cost of refactoring be lower than the gained benefits? How will the cost of refactoring change overtime? Its is all about the context of that particular TD. [2] In order to address this ambiguity the researchers from Chalmers University of Technology have developed an empirical model called AnaConDebt. With this model, the researchers strived to help practitioners determine if it is worth refactoring the ATD and also if it would be possible to determine how emergent it is to repay the debt. [4] Knowing this would be of great importance since the decision making and knowing when to implement certain features or paying back certain TDs will help increase the value of the product to a minimum cost. It is also identified as the best way to minimize the growth of TD, identifying the TD and what is causing it and thereafter managing them one by one [2]

The paper "In search of a metric for managing Architectural Technical debt", published at the 6th European conference on Software Architecture in 2012 stated that

if technical debt is not taken care of, it will lead to increased development costs, which will result in emerging technical issues related to the projects technology base.[5] Unfortunately, one the of most common ways of managing TD is to handle it ad hoc, meaning that it is taken care of when the crisis have already emerged. As can be expected this is not the optimal solution for managing TD hence the best time of refactoring has already passed. A major issue related to this is something called contagious technical debt. Contagious technical debt is a TD that overtime spreads across the system and is best managed by predicting its proliferation.[6] Recent systematic mapping research has stated that more tools with the aim of helping practitioners manage TD needs to be developed in order to make the process of keeping track of the TD part of the daily routine [11] This statement is strengthen by the theory that managing TD should be part of software development on the same level as implementing new features or fixing bugs[14][2].

The major research methodology used in this thesis is a design research. The purpose of the design research is to develop a software tool based on an ATD decision model, focused on the TDs in the red box in figure 1.1. This tool is to be evaluated by practitioners using ATD items that are identified in the system that is part of the investigation. The goal of this thesis is to investigate whether a tool of this kind would help software practitioners in their working life with increasing the software quality of their developed product. To be more precise, would the tool help them monitor their ATD items and take strategic decisions with respect to their future development in order to achieve a higher software quality. The research questions that is going to be answered in this thesis are,

RQ1a : When taking a decision on ATD, is there a difference between what aspects are used at the studied company and the ones suggested by the TD theory?

RQ1b :How can these aspects be combined?

RQ2 : How does an ATD decision tool affect practitioners in making refactoring decisions?

RQ3 : In what way does different stakeholders have different perceptions regarding ATD and its refactoring?

2
Background

In this section will the required background knowledge be described more in depth. The first part is about the environment for which this study is conducted. The second part is about the theoretical knowledge that the reader needs to be aware of for understanding what is being investigated in this thesis.

2.1 Domain background

This thesis is done in collaboration with a major product development company in the Gothenburg region active the automotive industry. The software system under investigation is an old system (development started in 1996) used for diagnostics of different hardware products. As a result of the software application being a rather aged system it has had several development cycles and iterations and therefore the system consists of several different programming languages, including vb.NET, C, C# and Java. During each life cycle different software design patterns have been used and promoted. Because of this there are certain parts of the system under maintenance just to keep the system working e.g. fixing critical bugs. This short term fix is mostly motivated by the fact that there are no business values gained from doing any major architectural changes, even though it might increase its overall software architectural quality. In the same manner the system is using executable files for which the source code no longer is available. To add even more complexity there are certain software components in the application developed by a third party developer.

2.1.1 Stakeholders

In this section the different stakeholders to the system under investigation will be introduced. Moreover, their role to the system will be described and potential individual benefits from this thesis will be presented.

2.1.1.1 Developers

The developers are the personnel who are developing the software system, including bug fixes, feature development and maintenance. The results from this research might not affect their work as they perform already approved changes by the business side of the company.

2.1.1.2 Architects

The architects in the domain are the stakeholders that have a higher level of knowledge about the systems software architecture. The results of this research especially has an effect on this group of stakeholders due to the fact that they are the ones who have the greatest knowledge of both the the current state of the software architecture and its ideal state. The tool would therefore help them by easily provide an overview of an Architectural Technical Debt (ATD) which would help motivate the refactoring decision for people on the business side of the company who has the final call.

2.1.1.3 Product Owner

The product owner are the ones who is in charge of the application under investigation and have the final word when it comes to decision making. This implies that any major refactoring needs to be approved by the product owner or other decision makers within the company with the same mandate before they can be executed by the engineering side. Hence, if the engineering side find a refactoring necessary, they need to be able to show and motivate its necessity.

2.2 Theoretical background

In this section the theoretical base of this thesis will be described in depth to help the reader acquire necessary knowledge and understanding. The sections will cover the current state of the art research regarding technical debt, architectural technical debt, models measuring architectural technical debt and tools for monitor technical debt and software architecture quality.

2.2.1 Technical Debt

As mentioned in the introduction Ward Cunningham was the first to express the term Technical Debt (TD) in 1992.[1] The meaning with the allegory is that solving issues fast, but with sub-optimal solutions, is similar to going into financial debt. The consequence of such decisions might be enhanced future costs. These postponed costs are equivalent to a financial debts interest. [8] To reinforce the technical debt metaphor, it is possible to state that the total debt is the distinction between the current solution and the optimal solution. [9] However, it is important to be aware of that not all technical debt is of a detrimental nature. There are situations where technical debt under control can be beneficial. [10] One example of such benefit is when a TD is continuously managed and under control from the beginning it can make it easier to reach the market faster. A positive consequence of this would be the possibility to get feedback from the customers in an early state, which then could be used to improve the product. [14]

When it comes to the cost of a TD there are a few important terms. The first one to understand is principal. The principal is the cost of doing the refactoring of

the TD today, in other words, paying back the debt immediately. In the same economical metaphor, the interest is the extra cost added to the software development due to the existence of this TD. [8] Martin Fowler identified four types of TD distinguished by the mentality of the ones who created that debt. This is also known as Martin Fowlers Technical Debt Quadrant and can be found in table 2.1. The factors in the quadrant are Reckless, Prudent, Deliberate and Inadvertent. Together these factors create a matrix of different implementation scenarios. In the first scenario a TD is created Recklessly and Deliberate since the implementers don't care about the outcome but still carries it out. In the second scenario it is created Recklessly and Inadvertently since they did not have sufficient knowledge in order to carry out that specific task. In the third scenario it is created Prudent and Deliberate as they are aware of the consequences but still continues since at that situation it is best choice. Lastly it is created Prudent and Inadvertent since they don´t have another choice at that time, but use the new knowledge acquired as a learning experience. [15]

	Reckless	**Prudent**
Deliberate	"There is no time for design"	"it needs to be released now, we deal with the consequence later"
Inadvertent	"What is MVC"	"Looking back, we know how to do it next time"

Table 2.1: Martin fowlers technical debt quadrant

2.2.1.1 Technical Debt Management

Recent study literary review (SLR) found that there are eight different TD management phases currently discussed in the software engineering community. These phases are,[11]

- TD Identification
- TD Measurement
- TD Prioritization
- TD Prevention
- TD Monitoring
- TD Repayment
- TD Representation
- TD Communication

Research has stated that the majority of software professionals had no default strategy in order to manage technical debt[12]. Research by (A.Martini, J.Bosch and M.Chaudron) reinforces this statement. They claim that the current way of managing the sub-category of TD, Architectural Technical Debt is ad hoc and is based

on the emerging of a crisis. Hence, the ATD is refactored when the optimal point in time to refactor has been missed. [6]

In contradiction to the aforementioned current way of handling refactoring, studies indicate prevention as one of the key phases in order to manage the TD of a software system. A recent SLR suggests a few counteractions in order to prevent the creation of TD. For instance, there should exist knowledge among the employees in order to minimize the TD that are created by mistake. Moreover, the development process should counteract the supervention of TD. [11] One development process that could counteract the creation of TD if used correctly is the Agile development process. This is due to the fact that it is about continuously improving software quality. Also suggested is that in order to prevent more TDs from being created the software architecture should be modularized. Furthermore, the usage of automated testing together with continuous integration is beneficial to minimize the risk of bugs being created. [16]

2.2.2 Architectural Technical Debt

Architectural Technical debt(ATD) is a sub-discipline of Technical debt(TD) which focuses on infringements on the intended software architecture. [6] ATD has also been defined as strategic decisions regarding the software architecture that affects internal software quality attributes.[11] One example of such quality attributes that the software architecture could be measures against are the standard ISO-9126. Recent research has investigated the emersion of ATDs and discovered that it depends on a number of different factors. These factors are stated below. [6]

1. Business Factors
   (a) Uncertainty of use cases in the beginning
   (b) Business evolution creates ATD
   (c) Time Pressure: deadline with penalties
   (d) Priority of features over product
   (e) Split of budget in project budget and Maintenance budget
2. Lack of specification on critical Architectural requirements
3. Reuse of legacy/third party/ open source
4. Parallel Development
5. Effects Uncertainty
6. Non-completed refactoring
7. Technology evolution
8. Human Factor

There has also been research that strived to classify different kinds of ATD items into different classes and map classes to the corresponding effect that indicates the presence of that type of ATD in the system. This mapping can be found in table 2.2. The different indications of presence are related to either the effort experienced by the programmer or the overall quality of the software being developed. For example, ATD of the type Dependencies unawareness could indicate that the ATD is contagious. This implies that there exist dependencies that are either not known to the developers or they exist where they should not. As a result of this the ATD spread across the system due to the creation of more unwanted dependencies. This also creates something called hidden ATDs, meaning that it is not always obvious that the ATD is spreading [6].

ATD class	**Indication of presence**
Duplication-reuse	Double effort and Repeated wrapping
Dependencies unawareness	big deliveries
Non identified quality requirements	quality issues
Non-uniformly policies	quality issues
Temporal Behaviour properties	confusion

Table 2.2: Types of ATD and their Indication of Presence

2.2.3 Architectural Technical Debt Analyzation models

When estimating a software project's technical debt, it is of major importance that the optimal level of software quality is defined and agreed upon with the stakeholder of the project. At the same time there must also be an acceptable level of software quality agreed upon. The software quality therefore needs to be between these two levels. [17] In the same manner there must also be a consensus regarding which software quality characteristics is the main focus for the application under development. Not all characteristics are equally important to every project. A commonly used standard for software quality characteristics is the ISO-9126.

The most frequently used way for measuring technical debt is using a calculation model that in most cases are relying on source code as input. Another approach for measuring technical debt is to relay completely on expert´s estimates [11]. There currently exists a number of defined methods for analyzing the software architecture quality, for example Architectural Trade-off Analysis Method(ATAM) and Scenario-based Architecture Analysis Method(SAAM). The SAAM method is used to verify that a systems software architecture satisfies both functional and non-functional requirements. Moreover, it is also possible to analyze risks with that architecture. ATAM is similar to SAAM in that it is mapping the software architecture to quality attributes in order to elicitate strengths and weaknesses. [22] However, there is also a newly developed model by Antonio Martini and Jan Bosch called the AnaCon-Dept model. While the previously mentioned methods SAAM and ATAM map and illuminate the architecture quality, this model takes well-defined ATD and suggests

whether or not it should be refactored. If the result of the model point to a refactoring, it also suggests when this refactoring should be done. Because of this distinct feature this is the model that will be used in this thesis and it will be explained more in depth in the next section. [4]

2.3 Technical Background

In this section, the technical base for this thesis will be presented. The section primarily focus on the AnaConDebt model, its construction and how it should be applied. The section will also cover the pre-existing tools for measuring software quality.

2.3.1 AnaConDebt: Construction and Appliance

AnaConDebt is a calculation model developed by Antonio Martini and Jan Bosch from Chalmers University of Technology. With this model the researchers strived to help practitioners take strategic decisions regarding the refactoring of Architectural Technical debts(ATD). To be more precise, to help the practitioners answering the questions if an ATD should be refactored, and if it should be refactored, when would it be more beneficial to conduct that refactoring[4].

In order to make any conclusions the costs for the ATD item first needs to be estimated in different time-perspectives. The total cost constitutes of two sub-cost, principal and interest. These two terms have recently been recognized as vital parts of TD management [11]. The principal is in economic terms the amount of money that was borrowed and agreed to be repaid when the loan was permitted. [24]. Converting the economical term into a technical debt term, the principal is the cost of repaying the technical debt today. It is estimated by calculating the costs of the two following factors,[4]

- The cost of refactoring the source of the ATD (Where is the source of the problem)
- The cost of refactoring all the related features/components which are affected by the presence of the ATD (Other parts of the system that are affected)

The second term, the interest, is in economic terms the cost of borrowing the principal[24]. This term means rather the same thing in technical debt terms. The Interest is the additional cost that are added due to the existence of the ATD and constitutes out of,

- Internal Propagation factors
- External Propagation factors

- Internal Impact
- External Impact

These four aspects can be divided into two categories, Propagation factors and Impact factors. Propagation factors are used to approximate how the ATD will grow and spread across the system over time if the debt is not repaid. [4] A TD that has propagation factors are also known as contagious technical debts. A table of the different propagation factors can be found in table 2.3

Internal Propagation factors	**External Propagation Factors**
Growth of the source of the ATD	Number of planned increments in the roadmap that will affect the ATD
Growth of the source code complexity	Number of external users

Table 2.3: Propagation factors

Impact factors are the additional cost of the interest. These so called Impacts can be further divided into Internal and External impacts. Internal impacts are costs related to the development and the maintenance of the product. External impacts are the cost experienced by the end users of the software. The currently identified Impacts are,

- Impact on Development Speed
- Impact on Maintainability
- Impact on qualities
- Impact on learning
- Non-completed refactoring
- Other Costs

The previously mentioned aspects are used to create these costs (Principal and Interest) for the different time perspectives Current, Short-term, Medium-term and Long-term. Using these perspectives or time intervals, two calculations can be done in order to help practitioners take strategic decisions regarding their ATD. The first calculation is to determine whether it is justifiable to refactor at all. Will the upfront cost of refactoring today be lower than the potential worst case cost in the future? Using the different time perspectives, it is possible to make a decision on when it is more beneficial to make the refactoring. The calculation for making the decision if an ATD should be refactored looks as the following,

$$\frac{Principal_{Current}}{Interest_{Total}} < 1$$

This calculation constitutes out of the two variables, $Principal_{Current}$ and $Interest_{Total}$. $Principal_{Current}$ is the cost of repaying the debt today while the $Interest_{Total}$ the worst case cost if the decision to refactor is postponed as long as possible. The Total time-span is usually based upon the Long-term time perspective. However, this is flexible and depends on how the time-spans have been previously defined. If the resulting value of the division is greater than 1 it means that cost of refactoring today dominates the cost of refactoring later. Therefore, there are no benefits cost wise to refactor today. In fact, it will be cheaper to do it in the future. On the contrary, if the value is closer to 0, it means that the cost of refactoring now is lower than the cost of refactoring later and then a refactoring should be done. The third alternative is that the value is very close to 1. This means that the cost of refactoring now will be roughly the same as in the long term. Hence, it is unclear (according to the formula) what is the best decision. In these situations, the final decision needs to be based on other aspects known to the domain experts. The the result from the calculation can then be used to support whatever decision made. The next calculation though is the calculation for deciding when a refactoring should be done and looks as the following,

$$\frac{Principal_{Current}}{Interest_{Total} - Interest_{Future}} - \frac{Principal_{Current}}{Interest_{Total}} < 0$$

The formula is a subtraction comparing the cost difference between two points in time, where the subtrahend is the same as for the calculating if a refactoring should be done at all. However, the minuend is the cost of refactoring in another point in time that is earlier then the $Interest_{Total}$ and later then the $Principal_{Current}$ from the subtrahend. If the result is less than 0 it means that the cost of refactoring today is larger than refactoring in the near future and it is therefore less convenient to refactor later. On the other hand, if the result is greater than or close to 0, it means that the cost of refactoring now or later will be roughly the same. Hence, it is not an emergency to refactor and it can therefore be postponed. However, if the difference is close to 0 there are a potential margin of error where the decision is not obvious. A more explicit figure explaining the AnaConDebt model can be found in figure 2.1

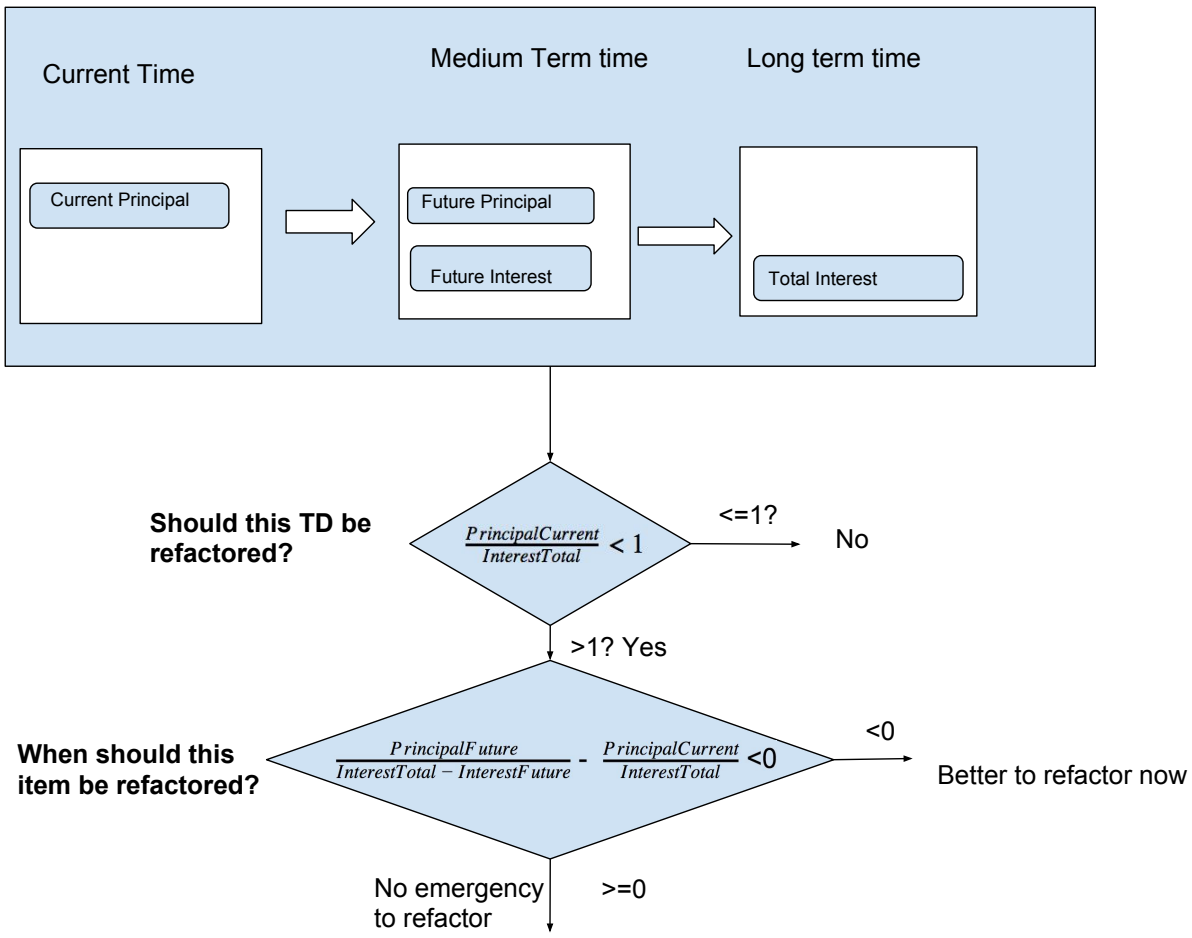


Figure 2.1: AnaConDebt method [4]

2.3.1.1 Generic Example of AnaConDebt

In this section a generic example of the AnaConDebt in practice will be showcased. The data used for this example is purely hypothetical and is not related to the company for which this thesis is conducted. For this example, the AnaConDebt model will be applied to an architectural technical debt(ATD) item called ATD item A and its data can be found in table 2.4. For simplicity only the time-spans Current, Short-term (6 months) and Long-term (18 months) will be used.

ATD Item	**Current Principal**	**Short-term Principal**	**short-term Interest**	**Total Interest**
ATD A				
Factors				
Cost of Refactoring the Source (CRS)	1000 (man hours)	1200 (man hours)		
Nr of releated features (CRI)	23	30	+7 features	+ 40 features
Growth of Complexity (GofC)			10%	30%
Impact on developer speed (IDS)			10%	30%
Impact on Maintainability (IM)			10%	40%
impact on qualities(how many?)(IQ)			2	3

Table 2.4: The data for ATD A

The principal, the cost of refactoring the ATD, is calculated by taking two factors in consideration. The cost of refactoring the source (CRS), which means the cost of refactoring the originated source of the ATD and the number of other features who are affected by the existence of the ATD (CRI). In order to determine when an ATD should be refactored it is necessary to estimate the principal for different time perspectives. In this case those time-perspective are Now (current principal) and Short-term principal (principal in six months).

As can be seen in table 2.4 the data is estimated using different scales. For instance, the CRS is estimated in man hours while number of relate features are estimated on the number of related features. The major question is how the values created using different scales are weighted against each other. The solution is to convert the scales so they are more compatible to each other. For instance, man hours can be converted into a ratio between the different time spans, where the current ratio is starting at 1. It is important to be aware of that these numbers are used for simplicity. Hence, in other cases these values also should have weight attached to them. As a result, the more important factors are valued higher. The CRS ratio for ATD A would therefore be,

$$\frac{shortTermCRS_{ATDA}}{currentCRS_{ATDA}} = \frac{1000}{1200} = 1,2$$

The CRS values for ATD item A used in this example can be seen in table 2.5.

ATD	Current Ratio	Short-term Ratio
ATD A	1	1,2

Table 2.5: CRS ratio values that can be used in AnaConDebt calculation

The same conversion needs to be done for the number of related features. As with the CRS, the difference between the time-perspectives is the interesting part.

$$\frac{shortTermCRI_{ATDA}}{currentCRI_{ATDA}} = \frac{30}{23} = 1.3043$$

The CRI values for ATD A can be seen in table 2.6.

ATD	Current Ratio	Short-term Ratio
ATD A	1	1,3043

Table 2.6: CRI ratio values that can be used in AnaConDebt calculation

The principal value for ATD A is then calculated by summarizing the CRS and CRI for the different time-perspectives. The equation for calculating the principal for these time-perspectives are,

$$Principal_{TimePerspective_{ATDA}} = CRS_{TimePerspective_{ATDA}} + CRI_{TimePerspective_{ATDA}}$$

The principals calculated using the equations above for ATD A can be seen in table 2.7.

ATD	Current Principal	Short-term Principal
ATD A	2	2,5043

Table 2.7: Principals values for ATD item A and ATD item B

As mentioned in section 2.3 about AnaConDebt the interest is the additional costs added to the principal as a result of the principal cost not being paid. Hence, the interest need to be calculated for specific time-spans. In this case the time-spans are Short-term and Long-term, where Long-term is called $Interest_{Total}$. This is due to the fact that it is the longest time-span that is taken in consideration for this case. These time-perspectives vary depending on who is defining them.

Just as with the principal there are a factor which needs to be converted to another scale. To be more precise, the factor "Number of related features" needs to be converted. It is important to recognize that the number of related features is in some contexts a principal and in some an Interest. This depends on whether the features are currently present in the system and are already affected or if the features

will be added to system. As can be expected the current value also start at 1 and the ratios are calculated in the same way. The ratios for ATD A can be seen in the calculation below. However, a summary of the values can be seen in table 2.8.

$$\frac{NrOfFeaatureInLongTerm_{ATDA}}{nrOfFeaturesShortTerm_{ATDA}} = \frac{30+40}{30} = 2,333$$

$$\frac{NrOfFeaatureInLongTerm_{ATDA}}{nrOfFeaturesCurrently_{ATDA}} = \frac{30+40}{23} = 3,043$$

ATD	Current Ratio	short ratio	Total Ratio
ATD A	1	2,333	3,043

Table 2.8: Summary table for the growth of the source for ATD A

The interests (Short-term and Long-term) for ATD A can now be calculated by adding all the factors which can be seen in the calculations below. However, it is important to be aware of that for these values the weights for each factor are overlooked. In a real case scenario these weights should be used in order to value the most important factors.

$$shortTermInterest_{ATDA} = 2,333 + 1.1 + 1.1 + 1.1 + 2 = 7,633$$

$$LongTermInterest_{ATDA} = 3,043 + 1.3 + 1.3 + 1.4 + 2 = 9,043$$

From this resulting values it possible to calculate whether an ATD should be refactored using the previously mentioned equation in the section 2.3 as following,

$$\frac{Principal_{Current}}{Interest_{Total}} < 1$$

The current principal is a combination between two different factors. One of these factors is the cost of refactoring the source (CRS), meaning the cost of refactoring the original source of the ATD. One example of this is when an API is not properly defined before it is used by other components. The CRS would therefore be the cost of fixing the API. The other factor that affects the current principal is the cost of refactoring the increment(CRI). Using the same example, this would be to refactor all the other components that are using the API in question. Using the previously calculated data it is possible to determine if those ATDs should be refactored.

$$\frac{Principal_{Current_{ATDA}}}{Interest_{Total_{ATDA}}} = \frac{2}{9,043} = 0,2211$$

The output from the previous division is the fraction 0,2211, which is clearly below one. This implies that the cost of refactoring the source(the dividend) is lower than the interest of not refactoring(the divisor). Hence, the cost of refactoring will be 4,5 times higher to do if the refactoring is postponed 18 months.

Now that it is determined that ATD A should be refactored, the question is when it is most suitable to do that refactoring. This question could be answered using the second equation in section 2.3 about AnaConDebt. The equations looks as the following,

$$\frac{Principal_{Future}}{Interest_{Total} - Interest_{Future}} - \frac{Principal_{Current}}{Interest_{Current}} < 0$$

Applying the previously produced data to the equation results in the following,

$$\frac{2,5043}{9,043 - 7,633} - \frac{2}{9,043} = 4,8981$$

As can be seen both the fractions results in fairly high values. This implies that it is more beneficial to refactor early. Hence, the ATD item should be refactored as soon as possible.

2.3.2 Tools for measuring code and software architecture quality

There currently exists a a number of different tools that strive to measure the software quality according to some predefined software quality metrics. The common approach for these tools are to use the technique static code analysis, or source code analysis. Two widely used tools of this kind are SonarQube and NDepend. There also have been attempts to develop tools that are more focused on the Software architecture quality. For example, Piyusa Meheshwari and Albert Teoh tried to develop a web based tool called ATAM Collaborative Environment(ACE) which incorporates the ATAM method though without any major success. [23].

3

Methods

In this chapter the methodology that was used to answer the research questions stated in chapter 1 will be explained. The thesis constitutes out of a design research which utilize two minor case studies. Section 3.1 focuses on the general knowledge needed for conducting a design research and section 3.2 focuses on the info needed for conducting case studies. Section 3.3 will then explain in depth how the design research and case studies are interconnected and how they are performed in the context of this thesis.

3.1 Design Research

Design research is a research methodology which focuses on solving problems by developing tailored software solutions. To be more precise, building an artifact(product) using state-of-the-art research from the research community in order to counteract the problem that the industry currently is experiencing. [13]

For this thesis the design research framework suggested by Alan Hevner, Salvatore March, Jinsoo Park and Sudha Ram is used. Conducting design research according to this framework constitutes out of a number of iterative steps. The first step is to understand the business needs of the industry, which is also known as the design research relevance. The second step is to gather knowledge from the knowledge base such as theories and models that are applicable from the research community. The third step is create an artifact (product or theory) that aims to solve the business needs using state-of-the-art theories and the fourth and final step is to evaluate the artifact using some-kind-of evaluation techniques such as experiments or case studies. This process is also known to assess the artifact and the outcome of the access process is then used to refine the artifact. This process is a cycle with no limits on the number of iterations, but at least one iteration is required. The resulting artifact is then released to the industry to solve their needs and the knowledge that are gained are publicized to the knowledge base. A figure that explains this flow can be seen in figure 3.1 [13]

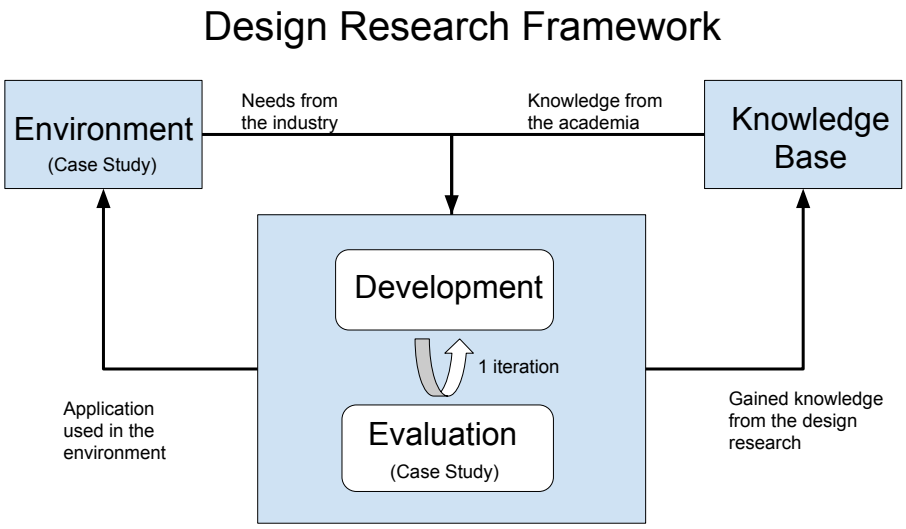


Figure 3.1: The Design Research Framework that is used in this thesis

3.2 Case Study Research

The usage of case studies as a research methodology are advantageous when factors, variables and its context continuously varies, making it a convenient choice in the field of software engineering. A case study consists out of several phases. First is the planning phase, where the guidelines on how to conduct the case study is set. Secondly the data collection phase, where the data for the case study is collected. The third and last phase is the analysis phase where the collected data is analyzed and dissected according to the rules specified in the planning phase.

For this thesis the frame of reference in how to conduct a case study is based upon the research of Runeson and Höst. In their paper "Guidelines for conducting and reporting a case study in Software engineering", they provide a number of bullet points of aspects that creates the foundation for the case study plan. These aspects are, [7]

1. Objective
2. The Case
3. Theory
4. Research questions
5. Methods
6. Selection Strategy

How these bullet points are used in the case studies are described more in depth in the case study sections 3.3.2.1 and 3.3.4.1.

3.3 The Design Research

As mentioned in section 3.1 the design research done in this thesis follow the framework suggested by Alan Hevner, Salvatore March, Jinsoo Park and Sudha Ram. The purpose of this design research is to develop and evaluate a tool which embodies the AnaConDebt model. For more information about AnaConDebt, see section 2.3. In the following subsections all of the different aspects of the research framework will be presented.

3.3.1 Knowledge Base

The first step when conducting this thesis was to find the current state-of-the-art research regarding technical debt(TD) and especially architectural technical debt (ATD). In order to do this a literature review was conducted and over 34 scientific papers and articles were read and analyzed. The selection of the read papers was done using the snowball principle which means starting with a few selected papers that were deemed good. Then, by going through their references, finding more in the same field, and so on until the base was complete [25] As previously mentioned in section 3.1 the purpose of this phase is to identify theories and models which can satisfy the issues discovered in the environment case study 3.3.2. The result of the literature review was to investigate, evaluate and apply the newly developed AnaConDebt framework to solve the business needs of the company as no such model had been used before. See section 2.3 for more information about AnaConDebt.

3.3.2 Environment

The purpose of the environmental aspect of the design research framework is to identify business needs in the industry that could be solved by the development of an artifact that incorporates some theory or methodology from the research community [13]. The question is therefore, how to identify the business needs of the company in this context. For this thesis it was deemed that the best solution was to conduct a case study.

3.3.2.1 Environment Case Study

As mentioned in section 3.2 a case study need to satisfy six aspects for the case study to be regarded as a valid. To begin with, the case study needs to have a clear objective. The primary objective for this case study is to identify the business needs for conducting the overall design research, but there is also the objective of identifying architectural technical debt(ATD) that currently lures in the system.

The second aspect is to have a clearly identified case. The case for this study is a software development project active in the automotive industry. The system in question is a widely used aftermarket system with over fifty thousand user´s world wide and is currently undergoing an investigation about a major refactoring to be planned in the near future. To be more precise, the software architecture will change

to be more modularized and therefore make it easier to rebuild or swap different modules.

The third aspect is to define which theory to use for the case study. The defined theory is based upon two theories. The previously mentioned case study structure suggested by Runeson and Höst in the paper "Guidelines for conducting a case Study in Software Engineering" and the theory behind AnaConDebt. See section 2.3 for an in depth explanation about AnaConDebt.

The fourth aspect is to have clearly stated Research questions to be answered by doing the case study. The research questions for this case study are,

1. What are the business needs regarding ATD?
2. Which are the major ATDs currently existing in the system?

The fifth aspect is the method for collecting data to answer the research questions. The method chosen for this study landed on preforming a number of semi-structured interviews with architects and senior developers. The interview structure followed the structure suggested by Runeson and Höst in the paper "guidelines for conducting a case study in Software Engineering". That the interview is semi-structured means that the questions varied between open and closed questions The interviews also followed the funnel principle, which implies that the order of the questions started by being more open and then going into more closed questions. The interview guide that created the foundation for the interview can be found in Appendix B.

The interviews constitute out of three phases. In the first and initial phase of the interview the interviewer explained the purpose of the interview and its usage areas. It was also of major importance to clarify the anonymity of the interviewee. When all that has been settled, the interview started by asking some open questions to get the conversation going.

The second phase was the main phase of the interview. This phase furthermore consisted out of sub-phases. The goal of the first sub-phase was to identify Architectural technical debt(ATD) items that the interviewee knew existed in the system. However, if the interviewee had difficulties in coming up with any applicable ATD items, leading questions where asked so that ATD items that where identified in previous interviews could be incepted to the interviewees mind. In the second sub-phase was more concrete and focused on the identified ATD items. In this phase the principal (cost of refactoring the source) and the interest (additional cost if the ATD item is not refactored) for the different time-spans today, short-term (6

months) and long-term (18 months) were estimated. The third phase for collecting data was to look through the systems backlog to identify already detected ATD items and documentation from previous TD investigations conducted by some of the architects.

The sixth and last aspect is the selection strategy for the case study. Meaning how the case was selected for this thesis. The case was selected because of the plan of a major refactoring of the system and that the collaboration company wanted to know which components could be saved. Since the objective for this case study was mainly to discover the business needs of the company and get as accurate values as possible for the AnaConDebt model people from all over the organization were used. The interviewees consisted of in total 10 people, ranging from developers, architects responsible for different parts of the system and the product owner. Seven of the interviews was done with people located in Gothenburg while three of where conducted over Skype with people in Curitiba, Brazil.

3.3.3 Development

The purpose of the development phase of the design research is to build an artifact or method that incorporates new theories from the research community in order to solve some business needs for the industry. For this thesis a tool(artifact) will be developed. This artifact is based upon an open source software tool which is released under the Apache v2 license. Unlike many other tools for managing technical debt(TD), this tool does not relay on using source code as input. Instead, the data is inserted by the users themselves. This software product is a web application that is built on ASP.NET Web API and Angular.JS 1.3. The main reason for selecting this repository as a foundation for the tool is that it is built on techniques that are approved by the company where the thesis is conducted. There are more criteria for the techniques that need to be fulfilled for it to be approved by the company. One of the more important reason for this is that the software technologies has reached a certain point of maturity. This requirement both ASP.NET and Angular.JS 1.3 achieves. Another reason that was taken in consideration while selecting this as a base was that the ease of platform independence since a Web application can be accessed by anyone as long as they have a functioning network access or running it locally on their own machine.

3.3.4 Evaluation

One of the most important aspects of conducting a design research is the validation part. Here questions like does the artifact fulfil the business needs that are specified by the customer, how does the artifact relate to the previously stated hypotheses. In the design research framework that is used for this thesis the techniques for evaluating the artifact or theory suggested are as following,

1. Experiment
2. Field Study
3. Case Study
4. Analytical
5. Simulation

It was deemed that a case study was the best choice as an evaluation technique. However, conducting an experiment was also taken in consideration, but eventually deselected due to the complexity of achieving validity in an experiment in this specific context. It would have required the experiment to be done in a closed environment were only the desired variables were changed in a controller manner. Because of the time constraints for conducting an experiment the most commonly used evaluation technique of Case study was therefore decided instead.

3.3.4.1 Evaluation Case Study

As mentioned in section 3.2 about case studies a valid case study needs to have six aspects covered. To begin with, it needs an objective. The objective for this case study is to evaluate how a tool that is built incorporating the AnaConDebt model is preforming in this context.

Secondly, it is necessary to have a clear case. The case for this case study is the same as for environmental case study in section 3.2. A software development project active in the automotive industry with over fifty thousand active users world wide. This system is also about to be refactored in order to achieve a better software architecture.

The third aspect is that it is required of the case study for it to be based upon some kind of frame of reference, also known as theory. The case study framework is the same as with the environmental case study, the case study framework suggested by Runeson and Höst. Moreover, the hypothesis is that an estimated refactoring decision is more accurate than others if it takes more factors in consideration. In the same manner there is a hypothesis regarding distribution of different kinds of factors which are considered when making a refactoring decision. Some factors are only considered in practice by the industry, some factors are only considered in TD theory and some factors are considered on both sides. Figure 3.2 visualizes this phenomenon.

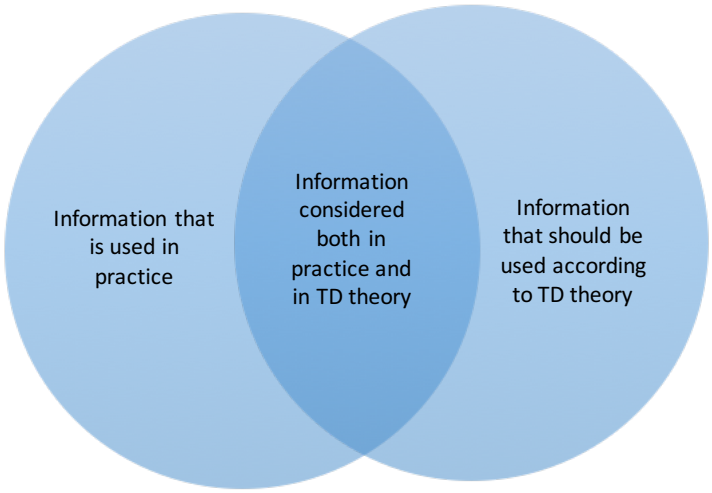


Figure 3.2: Venndiagram showing the relationship between factors used in practise and factors used in the theory

The fourth aspect is that it needs to have clearly defined research questions which will be answered by conducting the case study. The research question that is going to be answered in this case study are,

- Does the tool help in making an ATD refactoring decision?
- Which factors from the TD theory are acctually used in practise as well?

The fifth aspect is the method for collecting the data. In order to do this the tool is applied on one of the identified ATD items from case study 3.3.2.1 by practitioners who have a vast knowledge about the system and the nature of the ATD item. In total three people is going to participate, two architects and one senior developer. They are going to participate in each structured interview, which was a combination of practical work and answering a questionnaire. The questionnaire is designed so that the results can be easily quantifiable. The questions are out of four different types. Some questions yield yes and no answers while others the interviewee will have to give an answer on an ordinal scale between 1 to 5, where the extreme values represent the extreme opinions of the spectrum for the stated question. Another type of question are questions where the interviewee is asked to rank different aspects according to a ranking scale. The last question type of questions the interviewee is requested to list suggestions. For more in depth information about the survey, see

appendix C.

As mentioned in the objective of this case study this one focuses on evaluating the tool that is developed for this thesis. Hence, it is out of major significance to investigate how well the tool preform in making correct estimations regarding refactoring. In other words, there need to be some kind of comparison between two different decisions. One decision from the tool and one decision from another source. In this context the other source will be the top of the head knowledge by one practitioner who has vast knowledge about the projects software architecture. As a result of this the case study consists out of two phases. One phase where the interviewee will be given no aid except a whiteboard to write down his reasoning and one where he will use the tool. For both phases and all the different interviews, the same ATD item will be the object of study. The phases were limited to a maximum of 30 minutes each but before any of these phases could begin, was the interviewees were required to answer some general questions in the questionnaire. After that the process proceed with the phase were the interviewee only is allowed to use a whiteboard. After 30 minutes or if the interviewee feels done before, some questions related to the whiteboard phase were asked. A few examples of questions that were asked are,

1. Should this ATD item be refactored? (yes/no)
2. Which factors did you consider to take this decision?
3. If you had more time, which other factors would you have considered as well?

After this, the next step was the phase in which the interviewee was allowed to use the tool for the same ATD item. When the tool wizard had completed and the tool showing its output the interviewee had some time to analyze the result. After 30 min the interviewee were required to answer some tool and comparison between whiteboard related questions. A few examples of questions that were asked are,

1. Is the outputted suggestion from the tool different from your answer from using the whiteboard? (yes/no) a scale 1(not accurate at all)-5(very accurate), how accurate do you think the result of the tool was?
2. On a scale 1(not useful at all)-5(very useful), are the factors taken in consideration in the tool any useful?

An image showing the execution flow evaluating case study can be seen in the following image,

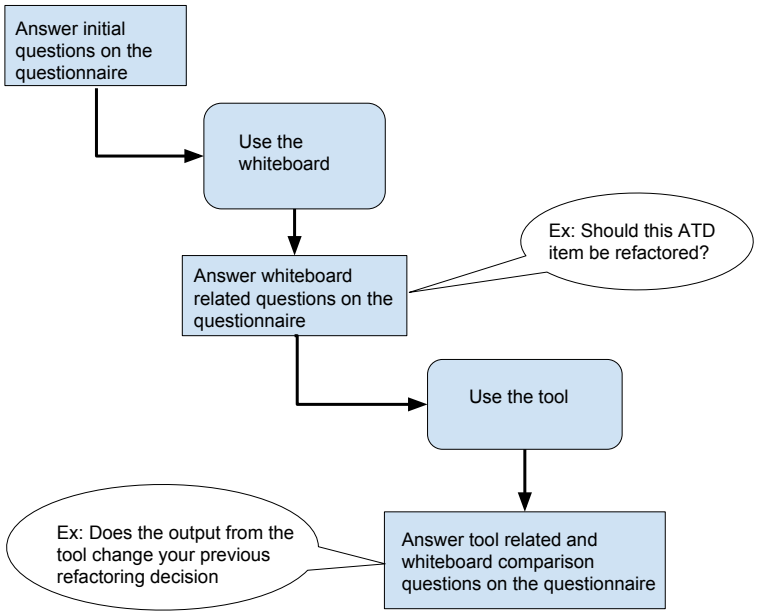


Figure 3.3: Execution flow of the case study for evaluating the design research artifact

The sixth and last aspect of a case study is the case study selection. This study was selected on the same reasons as the environment case study, the collaboration company suggested this case since it is a need for refactoring. As mentioned previously was one architect, one former architect and one senior developer selected for this evaluation. However, all of them had knowledge about the ATD in question.

4
Results

In this chapter the results of the design research are presented. The first part consists of the result from the environment case study and the second part consists of the result from the evaluation case study.

4.1 Environment Case Study

In this section are the results from the environmental case study stated. In this case study, the business needs of the company related to ATD was investigated. In the same manner was ATDs identified in their system.

The first step was to identify the business needs of the environment. The identified needs are as following.

1. prioritize future improvements
2. find the source and stop the spread of TD
3. reduce the current level TD and as a result build a better product quality wise

The outcome of the interviews was also a set of ATD items that currently exists in the system. The items that were collected and that were deemed appropriate in the context of architectural technical debt can be seen in table 4.1.

Identified ATD Item	**Number of Mentions**
Component had to much responsibility which should be extracted to its own component	5
Business logic were placed in wrong components	4
Unwanted dependencies between two components	1
Code standards are not consistent	2
Unwanted code duplication	1
Wrong software pattern which should be changed	2

Table 4.1: Identified ATD items from the interviews and how many of the interview they where mentioned

The table consists out of two columns. The first one describing the ATD item and the second one contains the number of interviews in which this particular ATD item was mentioned by the interviewees. The vast majority of the identified ATD items are relating to the software component which provides the system with its main functionalities.

4.2 Artifact Development

As previously mentioned the tool developed in this thesis aims to investigate whether a tool incorporating the AnaCoDebt model could solve the identified business needs of the company. See section 2.3 for more information about AnaConDebt and section 4.1 for more information about the identified business needs.

The tool is based upon an open source project which are released under the Apache v2 licenses on Github. The first thing that a user is prompted to do is to create a new Architectural technical debt(ATD) which are then added to a side menu. When an ATD item is selected, the editor view is displayed. When in editor view, there are two new major components showing, the toolbox and the canvas. In the toolbox there are a number of components which represents the factors from the ATD theory which are currently believed to affect the cost of refactoring an ATD. These factors can then be added to the canvas. In the canvas there are boxes representing the equations for calculating the cost of refactoring for different time perspectives. When all the equations have the right factor components and the right values, the user can press the calculate button which then takes the factor components and prints the result of the models decision (if and when the ATD should be refactored) together with a number of charts to motivate the decision. These charts are the number of estimated hours that would be needed to do the refactoring, how many other components that would be affected by the refactoring, the number of components that could be saved by doing the refactoring and the cost of refactoring and the interest at that time. This can be seen in the figure 4.1, 4.3, 4.3 and 4.4. However, the process of knowing which factor components to use for different time perspectives and which values and weights are quite time consuming. As a result of this the user can also choose to use the provided wizard. Images of the final version of the tool can be found in appendix D.

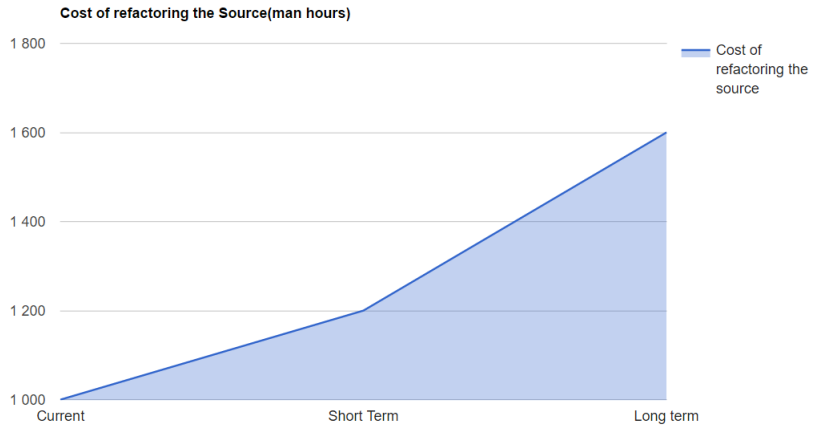


Figure 4.1: Man hours needed to conduct the refactoring

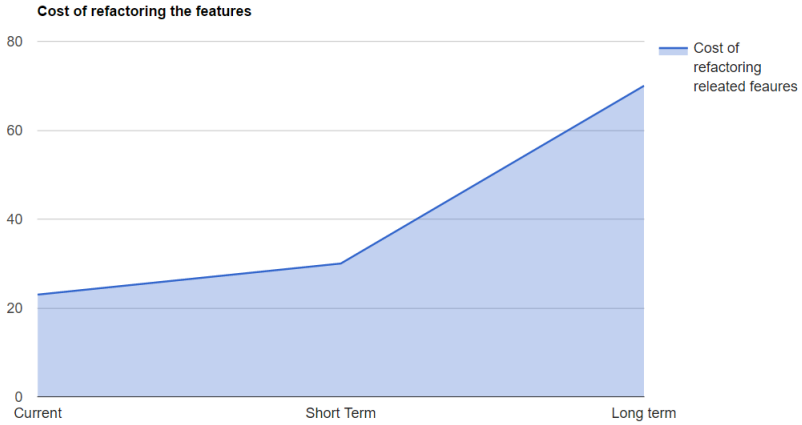


Figure 4.2: Number of components affected by the refactoring

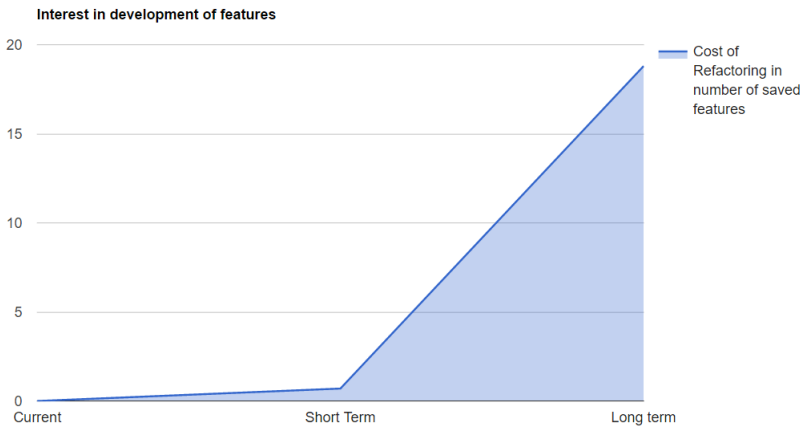


Figure 4.3: Number of components that could be saved from being affected by the refactoring if the refactoring is conducted

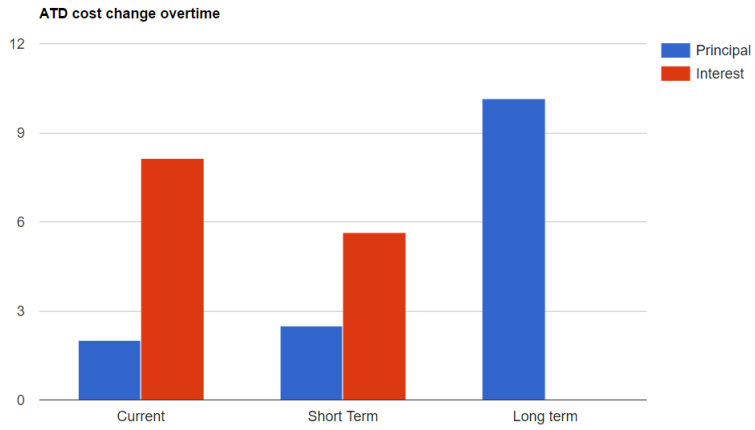


Figure 4.4: The cost of refactoring the ATD and the interest of not doing the refactoring for different time-perspectives

4.3 Evaluation Case Study

In this section the result from the evaluation sessions of the artifact of the design research is displayed. As mentioned in section 3.3 explaining the implementation of the evaluation are the main focus one of the identified ATD items from table 4.1. To be more precise, the ATD item that is used for this evaluation is called "Component had to much responsibility which should be extracted to its own component". However, for simplicity, this ATD is item from now on called ATD item A. This section constitutes out of three major parts, the questions that were asked before the interview started, the questions asked after using the whiteboard and questions asked after using the tool.

4.3.1 Questions before the session started

The initial questions in the interview focused on the participant and their current strategy to technical debt management and refactoring. The first question is about if there currently exists a strategy for deciding if something should be refactored. However, there was some ambiguity in their response. It was mentioned from two of the participants that there where no real strategy towards deciding if something should be refactored. The third participant contradicted this statement and expressed that a such a strategy did in fact exist. The same inconsistency appeared when the participants where required to answer if there currently exists a strategy for deciding when something should be refactored. Two participants expressed that it exists, while the third one said that it did not.

Another thing that was investigated was which factors that would affect the participant's decision for conducting a refactoring. In order to do so the participants were allowed to rank a number of different aspects that affect the prioritization of an ATD that were extracted from the published paper "Towards Prioritizing Architecture Technical Debt: Information Needs of Architects and Product Owners", by Antonio Martini and Jan Bosch. The following tables shows how the participants prioritized the different aspect. Table 4.2 shows prioritization between refactoring an ATD item and implementing a new feature and table 4.3 shows prioritization between two different ATD items.

Prioritization Aspects	**Person 1**	**Person 2**	**Person 3**	**Average**
Competitive Advantage	3	1	3	2.33
Specific Customer Values	2	1	3	2
Market Attractiveness	4	3	3	3.33
Lead time	3	3	3	3
Maintenance Cost	2	4	4	3.33
Customer long-term satisfaction	4	Not answered	3	Not applicable
Risks	4	5	5	4.66
Penalties	5	5	3	4.33

Table 4.2: Grading of factors to take in consideration when prioritizing between two items that need to be refactored. The lowest grade is 1 and represents that it is not prioritized while the highest grade is 5 and implies that it is very prioritized

Prioritization Aspects	Person 1	Person 2	Person 3	Average
Competitive Advantage	4	1	4	3
Specific Customer Values	3	1	4	2.66
Market Attractiveness	4	3	3	3.33
Lead time	2	3	5	3.33
Maintenance Cost	2	4	3	3
Customer long-term satisfaction	4	Not Answered	4	Not applicable
Risks	4	5	5	4.33
Penalties	5	5	3	4.33

Table 4.3: Grading of factors to take in consideration when prioritizing between item that needs to be refactored and a new feature. The lowest grade is 1 and represents that it is not prioritized while the highest grade is 5 and implies that it is very prioritized

From looking at the tables 4.2 and 4.3 it is possible to see that data is fairly consistent between participant 1 and 3. However, the responses of participant 2 has some outliners. For instance, participant 2 would not prioritize competitive advantage and specific customer values as he ranked them with the lowest score in comparison with person 1 and 2 who gave these two aspects the rank 4, the second highest. Also the grade Customer satisfaction was not provided by participant 2 while the others gave it a ranking of 4. Despite the differences between the participants it can be seen is that there is no major difference in prioritization between two items to be refactored, or between one item to refactor and implementing a new feature. Even though the values are not necessarily the same, if they where high in table 4.2 they where also high in 4.3 and vice versa. By looking at the data it is possible to draw the conclusion that the risk of doing the refactoring, the potential penalties, long-term customer satisfaction and market attractiveness was the most important aspects according to the participants.

All of the participants in the case study had a fairly good knowledge about the term technical debt, its meaning and its purpose. This is something that can be seen in table 4.5. In this case the lowest rank 1 is equal to "not at all" and the highest and 5 is equal to "very well". However, no one deemed themselves to be experts on the topic.

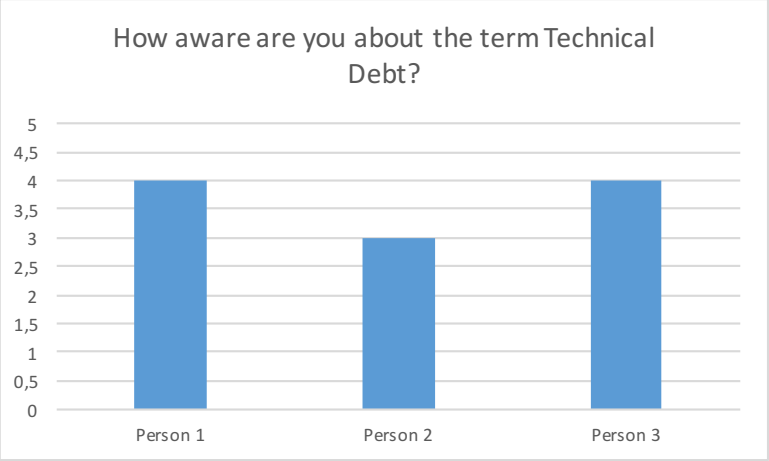


Figure 4.5: How much knowledge did the participants feel that they have regarding technical debt. 1 equals "not at all, while the highest score, 5 equals to "very well"

It was also inquired if there currently were any tools in use with the purpose of monitor the quality of the software architecture and how satisfied they where with it. All of the participants mentioned that they were using NDepend to monitor the source code quality. However, they did not have a fair opinion about it and could not express how satisfied they were with it since it had just been implemented to the project.

4.3.2 Questions after using the whiteboard

After using the whiteboard, the first question was if they would refactor that ATD item. This was a question that induced a wide spread of answers. One participant said that he would refactor this ATD, while another was quite unsure and said that it depended on the planned future of the component that inherited the ATD item. If the plan would be to make the component more generic, then he would refactor. However, if the component would fill the purpose as it currently possesses, he would not refactor since it is quite stable. The third participant also expressed that this ATD item is something that should be refactored. The participants were then encouraged to write down the factors that they used to make their refactoring decisions. Table 4.4 exhibits the different factors that each of the interview participants used in order to take the decision. As can be seen participant 1 considered five different factors, participant 2 six factors and participant 3 four factors. As can be expected some of the factors are the same, but with some difference in phrasing. The risk of refactoring was something that they all took in regard. Same as the costs of doing the refactoring and the potential gain.

Person 1	Person 2	Person 3
Risk	Risk	Risk
Value for Money	Future Business Value	Improvements (functional and non-functional)
Lead Time	Does it give an extra value	Maintenance
Maintenance	Amount of Business Value	Customer affects
Customer Satisfaction	Other Components	
	The code today	

Table 4.4: Factors that where considered on the whiteboard

After the whiteboard session the participants were asked whether they would have had considered any more factors if they would have had more time. Two of them expressed that they would do so, but none of them mentioned the same factors. One participant would look more into the complexity and another would look into if they really had a problem with the ATD today. To be more precise, if the component is stable even with the ATD, it may not be worth conducting the refactoring and thus increase the chance of making it unstable. The third participant expressed that he would not have considered anymore factors, even if he had more time to do so.

The final question to be answered was how accurate they thought their estimations was. The result of this can be seen in the figure 4.6, where 0 means "not accurate at all" and 5 means "very accurate". As can be seen it is a rather wide spread in how accurate the participants felt that their own estimation where. As can be seen in the following figure where the result where 4, 3 and 2 on a 1-5 scale.

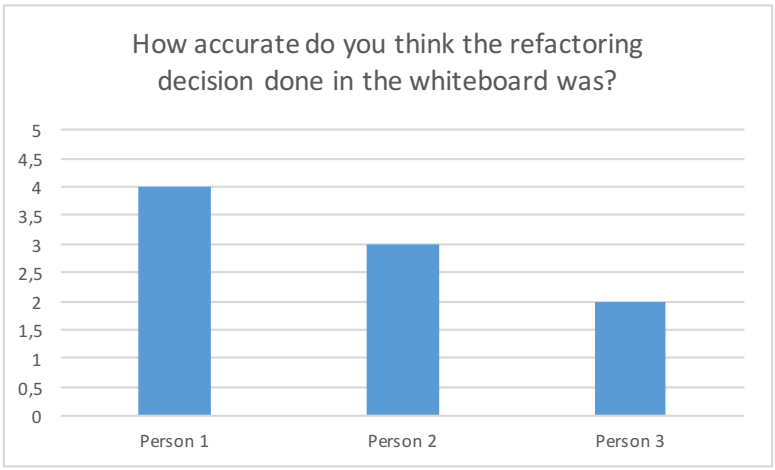


Figure 4.6: How accurate did the participants think that the refactoring decision by using the whiteboard was. The lowest score 1 equals to "not very accurate" while the highest score, 5 implies "very accurate"

4.3.3 Questions after using the tool

The first thing that the participants where supposed to look at after using the tool was the tools refactoring decisions and compare with their decision from previously using the whiteboard. Two participants did get the same decision as they previously came up with, that the ATD item should be refactored. However, the third participant did not get same as he had decided not to refactor while the tool suggested the opposite. However, the output from the tool did not change his mind and he would still not refactor.

After that the participants was supposed to estimate how accurate they thought that the output from the tool was. The result can be seen in figure 4.7 where 1 represents "not accurate at all" and 5 represents "very accurate". As can be seen all of the participants gave a 3 on this scale. Hence, it was regarded as sort of accurate. It was mentioned that the tool was sort of accurate with respect to the factor currently considered but in order for it to be more accurate should it should also consider the risk of doing the refactoring and as well as the components lifespan.

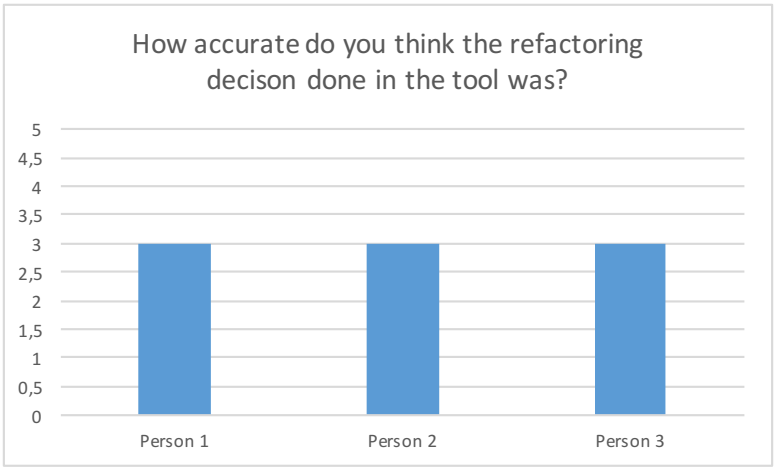


Figure 4.7: How accurate did the participants think that the output from the tool was. The lowest end of the scale 1 represents "not accurate at all" while the highest end of the scale, 5 represents "very accurate"

The factors that the tool was currently using were deemed rather useful as can be seen in figure 4.8, which utilizes the a scale on 1-5 were 1 represents "not useful at all" and 5 represents "very useful". One participant ranked the factors 4 while the other two participants gave them an average of 3.

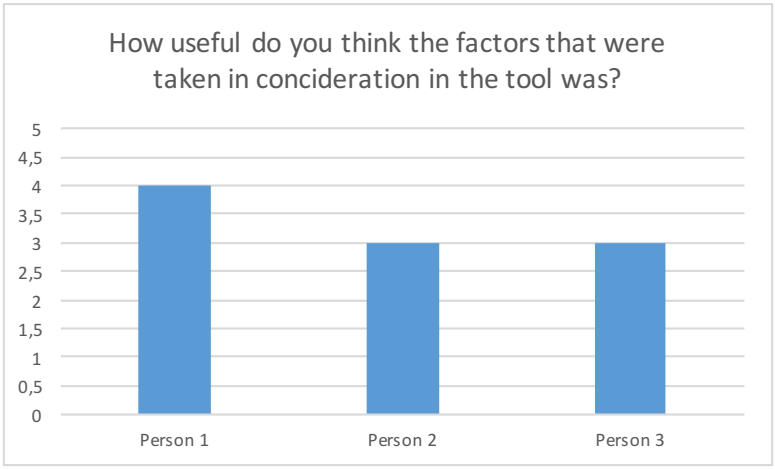


Figure 4.8: How useful did the participants feel that the factors used in the tool was. The lowest grade 1, represents "not useful at all" while the highest grade 5 equals to "very useful"

When it comes to the difference between the factors taken in consideration in the tool and on the whiteboard two of the participants felt that all of their factors should be used in the tool as well. Meanwhile the third one thought that only a few of them should be part of the tool. However, the implementation of the factors in the tool could have been fundamentally better. The participants also found that the output from the tool was not good enough since they did not think it showed the growth of refactoring cost rather well as can be seen in figure 4.9. As with the other diagrams it is based upon a scale 1-5 where 1 implies that it did not give any good feedback at all, while a 5 implies that the feedback was excellent. As can be seen this feedback lies around the lower end of the scale since the values where 2,1 and 3.

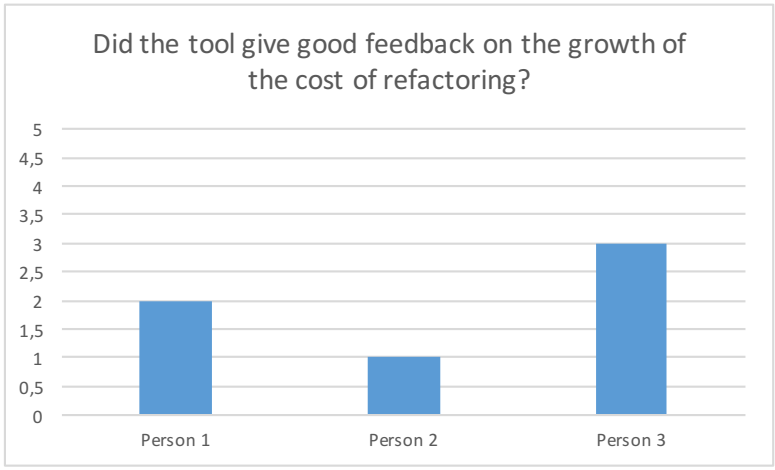


Figure 4.9: Did the tool give good feedback on the growth of the cost of refactoring. 1 is the lowest score and represents "the tool did not give any good feedback at all". The highest score 5 equals to "the feedback was excellent".

How the participants would use a tool of this type varied. One of the participants would use it for educational and informational purpose. Another one said that it

would be good for decision making together with architects and stakeholders on the business side if it would be possible to generate a report that could be attached in the backlog. The third participant would not use it at all, mainly because he had difficulties understanding the output from the tool. This is strengthened by looking at the following chart where the participant was allowed to grade on how much they felt that the tool would help communication with other stakeholders. As with previous charts implies it would not help at all, while a 5 would say that it helps a lot.

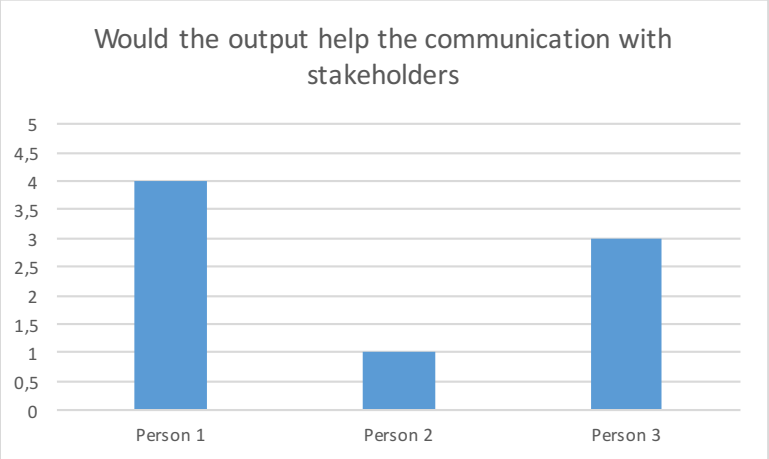


Figure 4.10: Would the output from the tool help communication with stakeholders. The lowest score 1, represents "the tool would not help communication with other stakeholders at all" while the highest score 5 equals to "that it would help a lot".

As can be seen in figure 4.10 one of the participant felt that the tool would help in the communication with other stakeholders while another did not see how this would useful for that purpose at all. It was also mentioned that the tool was not user friendly enough, but it could help communication with stakeholders if it used the right factors and presented them more clearly. The same can be seen in if they felt that the output from the tool would help them in the planning process when to decide what to do next. One participant did not see that this would help him at all while the others said that it would be useful with some minor modifications.

The final step was for the participants to give their feedback on how the tool could be improved. What was mentioned was that the participants wanted more quality attributes to be considered in the tool. It was also mentioned that the tool should consider the risk of doing the refactoring and the components life span. However, they also expressed that the tool gave insights about areas not thought of before.

5
Discussion

In this section the results from the chapter 4 will be discussed in order to answer the research questions stated in the chapter 1. Also the implication to the industry and academia will be debated. Lastly related work will be presented and potential future work suggested to guide research within this topic.

Recent research has identified factors which should be taken in consideration when taking a refactoring decision for an Architectural Technical Debt(ATD). However, the hypothesis used for this thesis is that there is a difference between the factors identified in the academia and those used in the industry. This hypothesis can be seen in figure 3.2. As can be expected not all the factors suggested from the academia, hence being incorporated in the tool, where used in practice by the industry. In the same way not all the factors used in practice where identified in the theory. Nevertheless, there was a union of factors. Using the whiteboard, the domain experts identified 5, 6 respectively 6 factors they would investigate in order to be able to make their decision. These factors can be found in table 4.4. From this is it possible to see that the number of factors taken in consideration by the domain experts were fewer than the factors used in the tool, which took 7 factors in consideration using the AnaConDebt model. More information about AnaConDebt can be found in section 2.3.

Figure 5.1 displays the relationship between the factors used in the whiteboard sessions and the the factors used in tool. On the left side of the figure all the factors used in the whiteboard sessions are presented. However, it is important to be aware of that this image is just a summarization. Hence, not all the factors that were used in practice where used at the same time. On the contrary, it is a merge of all the factors used by the participants during the three evaluation sessions. On the right side all the factors that are used in the tool are displayed and in the middle section all the factors that were used in both. The factors in this figure can either be yellow or white. All factors that were specific to either the TD theory or the industry is yellow. White factors imply that it was considered in both TD theory and in practice. Between the white factors, there are black and dotted grey arrows. The black arrows imply that the factors were a direct match to factors provided in the AnaConDebt theory. The dotted grey arrow indicates that the factor used during the white board session was approximately the same as one provided in the theory. In total were 11 unique factors provided during three whiteboard sessions,

7 factors where taken in consideration in the tool and only 4 were in common.

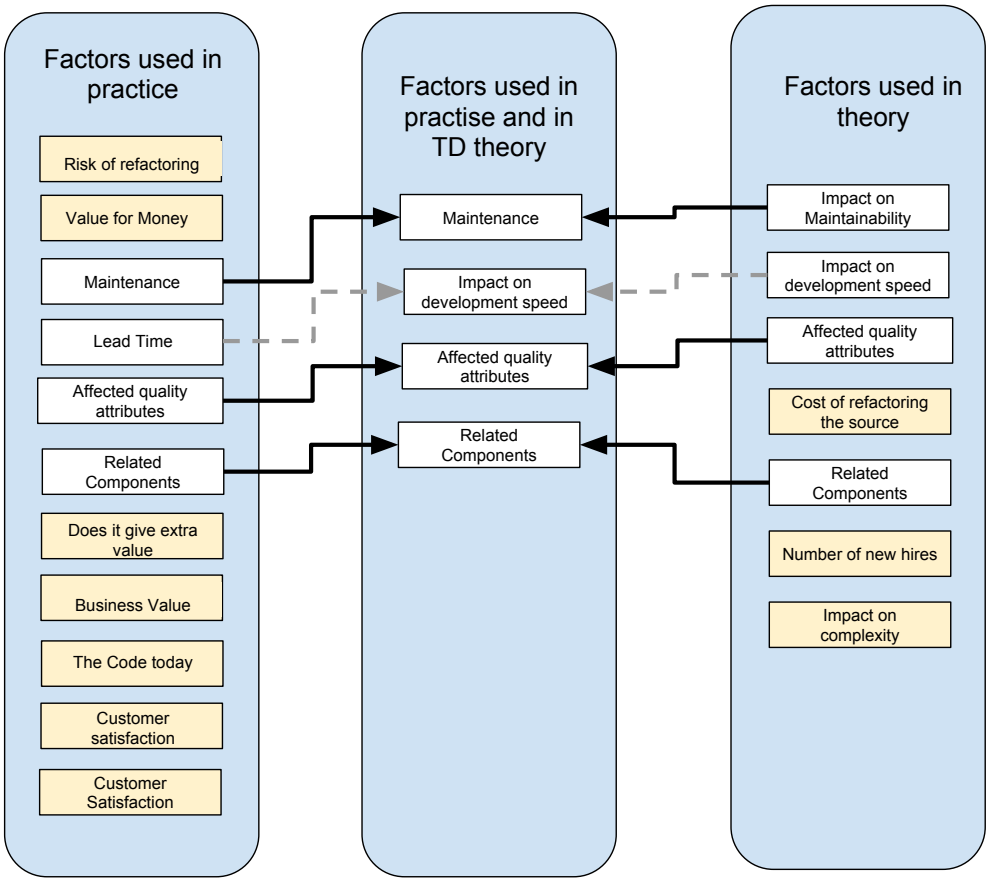


Figure 5.1: Diagram showing relationship between the factors used in practice and in TD theory. The factors from the theory are taken from the paper "An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt" by Antonio Martini and Jan Bosch [4]

5.1 RQ1a: When taking a decision on ATD, is there a difference between what aspects are used at the studied company and the ones suggested by the TD theory?

When making a decision regarding refactor, several factors are taken in consideration to determine if it is a good investment of resources. In order make such a decision, the academia suggest a number of different factors to be considered. It is interesting to investigate the intersections and differences between factors that are used in practice and those used in the academia. Looking at figure 5.1 only four factors

from TD theory was also used in practice. Those four were,

1. Impact on Maintainability
2. Related components
3. Affected Quality attributes
4. Lead Time/impact on development Speed

The question is, why did these four exist in the intersection between the TD theory and the domain knowledge in the industry? To fully analyze this, it is important to be aware of that there is a difference between the point of view of performing a refactoring between the industry and the point of view in the tool regarding risk. The tool is focused on the risks of not doing the refactoring compared to the industry which is more fixated on the risks of doing the refactoring.

Beginning with the impact on maintainability, which could be translated to the risk of bugs being created. It is quite obvious this was something that the industry felt was out of major importance. Hence, they do not wish to create more bugs by doing a refactoring of something that will not give any new features to their users. As mentioned previously the academia takes the other approach, what is the risk of bugs being created if the reconstruction is not done. Furthermore, both the TD theory and the industry recognizes the importance of knowing how big of an effort it would be to carry through the refactoring. They both also recognize the importance of knowing how inter-connected this component and its Architectural Technical Debt (ATD) with other components in the system. In other words, is the ATD isolated so that it will affect only a few other components or is it so complex that it would basically lead to a total reconstruction of the system? As the refactoring will not give any other features to the end users, it is important that they are given something else. This can be validated as both the industry and the academia considered the affected quality attributes. For instance, will the system be faster or more reliable after the refactoring? This is also related to the last factor in the intersection, the lead-time/impact on development speed. Will the refactoring lead to that future features can be developed faster and with less effort?

As mentioned before 4 out of 7 factors from the theory were used in practice. The theory specific factors specific were,

1. Cost of Refactoring the Source
2. Number of new hires
3. Impact on Complexity

Looking at these three factors, it might seem strange that at least cost of refactoring the source and impact on Complexity was not mentioned in any of the whiteboard sessions. One reason for this could be that the whiteboard sessions had a time restriction and making the participants forget to mention them. It could also be that they seem obvious and therefore making the participants forget to mention them. However, the number of new hires is not obvious. Will a postponed refactoring force new hires to learn the software architecture and the source code twice, before and after the refactoring?

5.2 RQ1b: How can these aspects be combined?

As mentioned in the section 4.3.3 two of the participants felt that all the factors they considered should be in the tool, and as a result the TD theory as well, while one only said that a few of them should. However, as mentioned in section 5.1 a few of the factors already exists in both theory and in practice as can be seen in figure 5.1. Looking at table 4.8 it is possible to see that the evaluators thought that the factors in the tool was useful. Likewise, from table 4.7 they thought that the decision from the tool was somewhat accurate with respect to the factors that it is currently using. However, it was mentioned that the model in the tool did not consider three key factors that all of the participants felt was out of major importance when deciding if something should be refactored. Those factors were,

1. The risk of doing the refactoring
2. The maturity of of the component which encapsulates the ATD
3. The life span of the component which encapsulates the ATD

To be more precise, is the component so vital that the risk of refactoring might lead to creating new bugs that out weights their current issue? This is the major difference between the practitioners and the academia. In the academia, the risk refers to the risk of bugs being created by not refactoring and for the industry it is the opposite. Another thing the practitioners felt that the theory should consider is the components maturity. Even though the component does not have the optimal software architecture from a quality perspective, is it stable with a few major features to be implemented that the ATD will not be a problem? And what is the plan for the component in the future? All of the participants suggest that these three factors should also be part of the model, and as a result, also the tool. Looking back, it is not feasible to validate whether or not all the factors that the participants considered on the whiteboard should be in the method. However, it is fair to say that the missing key factors should be included as they where mentioned explicitly by all three evaluators.

5.3 RQ2: How does an ATD decision tool affect practitioners in making refactoring decisions?

To begin with the participants felt that refactoring cost over time was not clearly stated. This can be seen in figure 4.9, where all of the participants graded it average or below, where one gave it the lowest score. However, this does not imply that such a tool is not useful. Instead, it proves that it is out of major importance to make the outcome of the tool more understandable. One of the participants saw potential and said it could help with decision making involving other architects and stakeholders. In the same manner two of the participants expressed that it could be used for informational purposes. One of the persons clarified that if the tool could generate a good report, it could be attached to the backlog and serve as background information. This is validated by looking at figure 4.10, where two of the three participants expressed that it could help them in their communication with other stakeholders. As can be seen the participants ranked the output of the tool to 4,3 and 1 on scale between 1-5. Hence, one participant did not see that it would help communicate with stakeholders. It was also deemed from two of the three participants that the tool could help them in the planning process of what to do next after some modifications to the tool. Examples of those changes are the missing key factors discussed in section 5.2. It was also mentioned from some of the participants that the tool gave them new insights in regard to what to take in consideration when making a refactoring decision.

What more can be seen is that the overall average value for the estimated accuracy for the whiteboard and the tool is the same. Looking at figure 4.7 all the participants estimated the accuracy of the tool to 3 on a scale between 1 and 5. However, looking at the accuracy of the decision done on the whiteboard in figure 4.6 each participant gave it a unique grade, 4, 3 and 2. Resulting in an average of 3. Looking at this is it possible to say that there was a more uniform belief in the tools accuracy compared to their own estimations. However, this does not imply that the tool was more accurate. The difference in the belief of the whiteboards sessions could be due to time constraints.

To conclude, does an ATD tool of this type help practitioners make refactoring decisions? There was a varied opinion about the tool as have been previously discussed. However, two of the three participants was more well disposed towards the tool while one was not. As the two participants who where more positive actually were software architects, which is the intended key users of this tool, it is possible to say that the tool would help making refactoring decisions. But it would need more development and to be more refined.

5.4 RQ3: In what way does different stakeholders have different perceptions regarding ATD and its refactoring?

In the evaluation process different attitudes did emerge regarding ATDs, its refactoring and a tool which aims to help making these decisions. In the evaluation process three participants were involved, one Architect, one former architect and one senior developer.

Starting with the tool, it was possible to see that the developer was more skeptical towards the usefulness of the tool. Even if the implementation was not perfect, both architects stated that the tool gave somewhat of an indication on the growth of the cost of refactoring, whilst the developer stated the opposite. Another distinction that was obvious between the different participants regarding the tool was how well it could be used for communicating about the ATD to other stakeholders. The architects gave it 3 and 4 on a scale between 1-5 while the developer gave it a 1(the lowest score). There are several reasons that could explain these distinctions. To begin with it could be that the tool shows quite a lot of information on the same time and was difficult to understand. Hence, the result might have been more in line if the graphical user interface was more intuitive and easier to grasp. Another explanation could be that the developer did not have the same perspective on the software architecture as the architects. Hence, the need to motivate a refactoring decision would not be a part of his working routine. The architect's role is to assess and question implementations, while the developer to some extent is the one responsible for fixing short comings in the code.

Another thing that was interesting can be seen in the figures 4.2 and 4.3. The numbers here are fairly consistent on how they would prioritize the factors which would affect their prioritization when choosing between two things to refactor. However, there are a few factors that varies quite a bit. For instance, the products competitive advantage or specific customer values. In this two areas two of the participants graded them average or higher on whether or not they were important to prioritize while the third gave it the lowest score.

To conclude, there were a vast difference on how practitioners approached an ATD refactoring and a tool incorporating such a decision model. The developer was continuously more skeptical while the architects saw some potential for its usefulness. Moreover, it is possible to say that there is a distinction in how developers and architects approach prioritization between features to implement and ATDs to refactor.

5.5 Implication for the Industry

Since it has been acknowledged by the research community in this field that a tool of this kind might help the industry make important decisions, this research is contributing by building and then examining the tool from user's different perspectives. As it turned out software developers and architects had different opinions regarding refactoring's of ATDs and its prioritization. This tool was in this context not useful for the senior developer and his working routine. On the contrast it could help the architects express the necessity for the people on the business side of the company who has the final call in making the decisions. This is due to the fact that the purpose of such a tool is to make something that is obvious to them with their level of domain knowledge visible and understandable for non-technical people.

5.6 Implication for the Academia

The implication for the academia is that from the results of this thesis it could be concluded that their model AnaConDebt is not quite there yet for it to be useful in the industry. From this evaluation was three additional factors emerged as important to consider,

1. The risk of doing the refactoring
2. The maturity of of the component which encapsulates the ATD
3. The life span of the component which encapsulates the ATD

Moreover, it is stated from the intended key users of such a tool, the software architects, that there is a potential future for such a tool in the industry. Hence, there is a motivation from the industry to continue research on this topic. One implication to this, that also was not obvious before, is that in this context there is a difference in the mindset towards ATD between software architects and developers. This is therefore something that needs to be considered in future research.

5.7 Limitations

It is important to be aware of that this is a design research. Thus, the goal of the provided tool is not to be a tool that solves all of the business needs from company. On the contrary, the aim is to solve some of their business needs using state-of-the-art theories. In this case, the recently developed AnaConDebt model. Hence, the provided artifact is a proof-of-concept of a tool that incorporates an Architectural technical debt (ATD) decision framework. Thus it is more about evaluating the concept of such a tool rather then build the perfect tool at this moment. Moreover, all the data provided in this thesis is coming from only one company. The opinions regarding the pros and cons of the tool is coming from only one certain type of company and does not necessarily reflect the absolute truth. Hence, the opinions

might be different if the research would have been performed at other companies as well.

5.8 Threats to Validity

In this section different types of threats to the validity of this thesis will be discussed. In the context of this thesis the threats are divided into four sub-types as suggested by Runeson and Höst [7].

5.8.1 Construct Validity

To begin with there are some issues with construct validity in both in the environmental case study 3.3.2.1 and in the evaluative case study 3.3.4.1. In both of these interviews were conducted which opens up for misinterpretation between the interviewee and the interviewer regarding the asked questions. Moreover, both case study interview sessions involved questions asking for estimations. It is not possible for the researcher to validate these estimations. It is therefore possible that the participants gave more optimistic estimations to make the situation look better than it actually is or vice versa. However, in order to minimize this effect, the researcher ensured the participant that all the data where totally confidential. Furthermore, three people were involved in this process in order to triangulate the results.

5.8.2 Internal Validity

One Internal threat to validity is present in the evaluation case study in section 3.3.4.1. This threat is related to whether the interviewee already had made up is mind before hand and only mentioned factors to strengthening this reasoning and avoided those who would contradict this. This is almost impossible to avoid due to to the fact that the researcher strived to be unbiased and as a result did not mention factors that other participants had used.

5.8.3 External Validity

This design research was conducted with only one company and focusing on only one of their systems. In the same manner the evaluation was done only with parts of one of the team involved in the development of that particular system. This is an external validity threat. Hence, if there would have been more time, involving people from other teams would have been preferable in order to further triangulate the results. It would have been even more beneficial to involve several companies in this research. Due to this uniformity, it is not obvious that the results would have been the same if more people was involved.

5.8.4 Reliability

There exist one obvious threat of validity regarding the reliability of this research that is present in the evaluation case study described in section 3.3.4.1. When

conducting interviews, there is a possibility that the researcher might influence the participants reasoning. This is obviously something that is almost impossible to avoid. However, in order to minimize this effect, the researcher strived to avoid unnecessary conversation with the interviewee during the session. Some conversation was needed though in order to explain what the participants should do and clarify if the participant had difficulties understanding the task.

5.9 Related work

It is a common practice to use some kind of mathematical model in order to measure and quantify the software quality of system. A common approach is to use tools that incorporates models and rules which is based on inaccuracies on the actual code. Examples of such tools are SonarQube and Ndepend. However, the process of measuring architectural technical debt (ATD) is a bit more complicated and resource demanding. A common method for these kinds of tasks are the Architecture Trade-off Analysis method(ATAM). ATAM constitutes out of four major phases. These four phases are described in broad terms below, [26]

1. Presentation:
   In the beginning of this phase the ATAM method is described to the stakeholders. After that the business goals that motivates the necessity of this evaluation are presented by the project manager. Lastly the suggested software architecture that aims to satisfy the business needs are presented by the architects
2. Investigation and Analysis:
   In this phase different approaches to achieve the suggested architecture are identified. The quality factors that are currently affecting the system in a negative manner are collected and the different scenarios where these are an issue are also analyzing. The highest ranked quality factors are then mapped to the architectural approaches. These approaches are then further analyzed to discover strengths and weaknesses.
3. Testing
   This phase is rather similar to the previous phase with the difference that all the stakeholders are present. Different scenarios are identified that are affecting the quality attributes and are then prioritized. The highest prioritized scenarios are then used as test cases for the identified architectural approaches.
4. Reporting:
   In this last phase the ATAM team presents the result of the evaluation. They can also conduct a report of the findings and suggested methods to shift to the new architecture.

However, this process is rather time-consuming and requires a lot of participants [21]. The basis for the ATAM model and the AnaConDebt model used in this thesis is rather similar. Both takes an ATD and strives to estimate a cost of repaying that debt. The major difference is that AnaConDebt takes it a step further and tries to

answer if the debt should be repaid and when refactoring should be done compared to ATAM which shows the cost and how it is related to specified quality attributes. There has been attempts to build tools upon these models. For example, in 2005 Piyush Maheshwari and Albert Teoh tried to build a tool based on ATAM called ATAM Collaborative Environment (ACE). However, it was unsuccessful and was never completed. It was mentioned that the major issue with the tool was the make it easy to do an ATAM evaluation interacting with a computer [23]

5.10 Future Work

Regarding potential future work there are several aspects that could be looked into. To begin with it would be interesting to investigate how to re-engineer the AnaConDebt model and a corresponding tool so that it would also assess the risk and stability of the system in consideration. It would also be interesting to build a tool even more focused on the user experience in order to make it more user friendly and as a result might lower the difficulty threshold. One area that should be further investigated related to ATD is the sensitivity in extracting information about these items. This is because knowledge about certain points in the architecture might lead in the direction of different interviewees. It would be interesting to investigate how to minimize this effect since the lack of transparency leads to completely different outcomes of an architectural assessment. This is also regardless of tools and models. Lastly, it would be interesting to investigate how to integrate the ATD tool with the actual refactoring. In other words, how should this tool be used in collaboration with the architects in their daily work.

6
Conclusion

Today, it is becoming more and more expensive to develop software. The buyers require a higher quality of the software as their own knowledge regarding software increases and they often wish to have a leverage towards their market competitors. In the same manner it is often a desire that the software is easily maintainable in order to be able to keep this advantages in the future. To address this issue software companies have started using agile development methodologies which encourages the developers to be alert and flexible to changes, compared to the waterfall model where the requirements are locked before the development begins.

The usage of agile methodologies encourages the necessity of frequent refactoring and as a result of this it is not only required to know what should be refactored, but also, it is feasible to do it from a cost-perspective. Would the time required to do the refactoring result in gained value that out weights the cost of doing the refactoring. In order to simplify this Ward Cunningham coined the metaphor technical debt(TD), which solves this using the economical concept of debt. In this metaphor all implementations that are not optimal represent a debt which has an interest. The interest could be described as the additional cost of having this debt in the system. The TD notion contains a subset of sub TDs and the focus in this report is the one called Architectural technical debt (ATD). ATD is architectural inaccuracies, meaning that the software architecture is currently implemented in way that is sub-optimal according to its ideal state.

The process of analyzing the ATD is currently quite time-consuming and requires a lot of resources. One example of an established method for measuring the quality of software architecture is the Architecture Trade-off and Analysis Method(ATAM). In ATAM, risks in the software architecture are mapped to different quality attributes which then are compared to each other in order to find weaknesses that should be focused on during the development. However, in this thesis, an alternate model is investigated, the AnaConDebt model. In this model an ATD is analyzed in order to help practitioner take the decision if it should be refactored and when that refactoring should be done. Recent research has also expressed the necessity of tools for these kinds analyzes. In this thesis a tool was developed to investigate this using the AnaConDebt model. The tool developed was incorporating the AnaConDebt model and was evaluated with a system at a company that is planned to undergo refactoring in the near-future. In this thesis four research questions were then investigated

and the findings were,

- **RQ1a: When taking a decision on ATD, is there a difference between what aspects are used at the studied company and the ones suggested by the TD theory?**
  In total four factors from the ATD theory were also used in practice. Those four where,
  1. The impact on maintainability
  2. The components related to the ATD
  3. How different quality attributes are affected
  4. The impact on development speed.
- **RQ1b: How can these aspects be combined?**
  Three factors used in practice should according to the interviewed industrial professionals be added to the theory. Those three where,
  1. The risk of doing the refactoring
  2. The maturity of of the component which encapsulates the ATD
  3. The life span of the component which encapsulates the ATD
- **RQ2: How does an ATD decision tool affect practitioners in making refactoring decisions?**
  This kind of tool does help architects communicate the severity of an ATD to other stakeholders and make refactoring decisions. However, the tool needs some improvements, for example regarding user friendliness.
- **RQ3: In what way does different stakeholders have different perceptions regarding ATD and its refactoring?**
  There is a vast difference between architects and developers in their approach towards ATD, refactoring and ATD refactoring decision tools. The developer was in general more skeptical while the architects saw potential.

From this it is possible to see that some of the most vital factors from the ATD theory are in fact used in practice as well but there were three other factors suggested by the industry that should be added to the AnaConDebt model. The industry needs it to consider the risk of doing the refactoring, how mature the component is and the planned road map for the component. Another interesting discovering is that there is clear difference between software architects and software developers in their approach towards ATD and its refactoring.

In summary, in this thesis was a proof-of-concept of an ATD decision tool developed. It was deemed from the intended key users, the architects, that it had potential to help make a refactoring decision for an ATD and spread information to other stakeholders. However, it needs improvements for it to be useful. If the tool is made more simple to use and the ATD model it relies on considers more factors a tool of this type could help the industry take the next step in solving the problem of ATDs that are now current.

Bibliography

[1] Cunningham, W. (1992) The WyCash Portfolio Management System

[2] Kruchten, P., Nord, R. L., Ozkaya, I. (2012). Technical debt: from metaphor to theory and practice. IEEE Software, (6), 18-21

[3] A. Martini and J. Bosch, “The danger of architectural technical debt: Contagious debt and vicious circles,” in Software Architecture (WICSA), 2015 12th Working IEEE/IFIP Conference on, May 2015, pp. 1–10.

[4] A.Martini and J. Bosch, "An Empirically Developed Method to Aid Decisions on Architectural Technical Debt Refactoring: AnaConDebt" (2016)

[5] Robert L. Nord, Ipek Ozkaya, Phillippe Kruchten, Marco Gonzalez-Rojas, "In Search of a Metric for Managing Architectural Technical Debt" (2012) Joint Working Conference of Software Architecture 6th European Conference on Software Architecture.

[6] A. Martini, Jan Bosch and Michel Chaudron, "Architecture Technical Debt: Understanding Causes and a Qualitative Model"(2015)

[7] P.Runeson and M.Höst, "Guidelines for conducting and reporting case study research in software engineering" Empir Software Eng (2009) 14:131–164

[8] http://martinfowler.com/bliki/TechnicalDebt.html, (2003),Martin Fowler

[9] A.Nugroho, J.Visser and T.Kuipers, "An Empirical Model of Technical Debt and interest" (2011)

[10] E.Allman, "Managing Technical Debt - Shortcuts that save money and time today can cost you down the road.", (2012), ACM 55(5), 50-55

[11] Z.Li, P.Avgeriou and P.Liang, "A Systematic mapping study on technical detb and its management" (2014) The Journal of Systems and Software 101(2015) 193-220

[12] N.Ernst, S.Bellomo, I.Ozkaya, R.Nord, I.Gorton "Measure it? Manage it? Ignore it? Software Practioneers and Technical Debt", (2015)

[13] Alan R. Hevner, Salvatore T. March, Jinsoo Park and Sudha Ram, MIS Quarterly Vol. 28, No. 1 (Mar., 2004), pp. 75-105, “DESIGN SCIENCE IN INFORMATION SYSTEMS RESEARCH”

[14] P.Krutchen, R.Nord, I.Ozkaya and D.Falessi "Technical Debt: Towards a Crisper definition" Report on the 4th International Workshop on Managing Technical Debt, (2013) September, volume 38 number 5

[15] M.Fowler, "http://martinfowler.com/bliki/TechnicalDebtQuadrant.html", (2014) 20 november, accessed 2016-10-08

[16] D.Radigan, "https://www.atlassian.com/agile/technical-debt", (2016)

[17] J.-L. Letouzey, "The Scale Method for evaluating Technical Debt" in Proceedings of the Third Internationa Workshop on Managing Technical Debt, Piscataway, NJ, USA (2012), pp 31-36
[18] Patrick Smacchia, Improve your .NET code quality with NDepend, "http://www.ndepend.com", (2004), accessed 2016-07-13
[19] SonarQube, http://www.sonarqube.org/, (2008), accessed 2016-07-14
[20] "Software Quality Metrics", http://www.sqa.net/softwarequalitymetrics.html, accessed 2016-09-19
[21] "http://www.sei.cmu.edu/architecture/tools/evaluate/atam.cfm", Software Engineering Institure, Carneiege Mellon, accessed 2016-10-09
[22] Liliana Dobrica, Eila Niemelä, "A Survey on Software Architecture Analysis Methods", (2002)
[23] Piyush Maheshwari, Albert Teoh, "Supporting ATAM with a collaborative Web-based software architecture evaluation tool", (2005)
[24] Consumer Financial Protection Bureau, "What is the difference between paying interest and paying off my principal in an auto loan?" "http://www.consumerfinance.gov/askcfpb/845/what-difference-between-paying-interest-and-paying-my-principal.html", accessed 2016-10-10
[25] Snowball versus Respondent-Driven Sampling, Douglas D. Heckathorn https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3250988/, accessed 2017-01-08
[26] R.Kazman, M.Klein, P.Clements, "ATAM: Method for Architecture Evaluation", (2000)

A

Interview guide

A.1 General Information

A.1.1 Goal with the Interview

- Gain general information about the system
- Identify ATD items
- Get principal and Interest for the identified ATD items.

A.1.2 Interview Type

- Semi-structured interview
- Funnel-model(beginning with open questions and then becoming more specific)

A.1.3 Interview Phases

1. First phase
    - present objectives of the interview and case study, how will the data extracted from the interview will used, get permission to record the interview, ask some simple introduction questions
2. Second phase
    - main interview questions
3. Third phase
    - summarize the major findings, and make the interviewee confirm that it is correctly understood, identity misunderstandings and possible feedback on the questions/answers or the structure of the interview.

A.2 Guide

A.2.1 General info to establish with the interviewee before the interview starts

- Explain the topic, ATD, What I wish the get from this interview, what the outcome will be and that they are anonymous
- Let them describe themselves, what is their role, how long they have been working with system etc

- Let them describe themselves, what is their role, how long they have been working with system etc

A.2.2 ATD Identification

A.2.2.1 Open start questions

- In which components in the system have you been working on the last year?
- Which are the five most complex areas in the system to make software related changes?
- Are there parts of the architecture that you are considering to be sub-optimal?
- What are the extra-cost that occurs due to this?
- What are the worst architectural parts of this system?

A.2.2.2 Factors to Identify

Goal: Identify code duplication
Explanation: Identify presence of similar code in different parts of the system, managed separately and not grouped into reusable components.
Example Question:

- Do you have any unwanted double maintenance (at least two components that are similar, but with small changes)

Goal: Identify unwanted dependencies
Explanation: TODO: Skriv något snyggt här
Example Question:

- Do you have any unwanted ripple effects? (meaning that a change in one place, leads that you need to change somewhere else that was not intended with the architecture)
  - Could this be because of unwanted depencies?

Goal: Identify unwanted architectural patterns(patterns and policies that are not kept consistent)
Explanation: TODO: Skriv något snyggt här
Example Question:

- Is it difficult to understand the system?
- What architectural patterns currently exist in the system?
  - Any of them you want to remove?

Goal: identify non-functional requirements that were not taken in consideration at the beginning of the development
Explanation: TODO: Skriv något snyggt här
Example Question:

- Give examples on non-functional requirements. Are you experiencing issues regarding these requirement types?
- Do you have any issues with software quality?

Goal: Identify temporal properties of interdependent resources
Explanation: concurrent and non deterministic interaction with the resource by different components. EX: convention of only having synchronous calls to a certain component. However, one of the teams used forbidden asynchronous calls.
Example Question:

A.2.3 Principal

Goal: Find the cost of refactoring the debt(refactoring away the ATD) ex in man hours
Explanation:
Example Question:

- what do you need to do in order to refactor this item?
- how much time do you estimate that it will take? (man hours)

A.2.4 Interest

Need to determine values for short and long term.

A.2.4.1 Propagation Factors

Factors that causes the ATD to propagate to other parts of the system

A.2.4.1.1 Internal factors **Goal:** Find the number of functionalities that will involve the ATD in the chosen lifespan
Explanation: if the ATD was included in a single component, estimate how many new functionalities would be included in the component in the chosen life span(short,medium,long)
Example Question:

Goal: Estimate how the complexity will grow over the chosen life span
Explanation: as a multiplier
Example Question:

- How much more difficuly will it be to develop something due to the existence of the ATD?

A.2.4.1.2 External Factors **Goal:** find number of increments in the roadmap that will include the ATD
Explanation: for those features that will involve/interact the ATD, estimate a refactoring cost for those features as well. (man hours)
Example Question:

Goal: number of external users in the system?
Explanation:
Example Question:

- How will the usage of the tool increase/decrease in the chosen lifespan?

A.2.4.2 Impacts

Goal: Find the impact on development speed(%)
Explanation:
Example Question:

- Use the propagation factor number of increments(roadmap), how much will the development teams spend in extra effort?

Goal: find the impact on maintainability ((%)
Explanation:risk of bugproness and the consequences extra cost in fixing the bugs. (ex overhead is growing with 10
Example Question:

Goal: find the impact on quality (ISO-9126)
Explanation:give a list of qualities, ask them which are more important. How would an such quality be affected?
Example Question:

Goal: find the impact on learning
Explanation:if the refactoring would be postponed, new hires would need to learn the system twice.
Example Question:

Goal: Impact in revenues
Explanation: waiting to refactor would block the possibility of selling features separately.
Example Question:

B

Design Research - Case Study Evaluation

B.1 Questions to be answered before the session starts

- On a scale 1(just began) -5(extremely familiar) how much experience do you have with the project?

- Do you currently have a strategy for deciding what to refactor in your project(yes/no)

- Do you currently have a strategy for deciding when something should be refactored(yes/no)

- Rank the following aspects, 1(not important at all) - 5(extremely important), which of them are important to take in consideration when prioritizing between two items that needs to be refactored, but only one can be refactored right now?
  - Competitive advantages
  - Specific customer values
  - Market attractiveness
  - Lead time
  - Maintenance cost
  - Customer long-term satisfaction
  - Risks
  - Penalties

- Rank the following aspects 1(not important at all-5 (extremely important), which of them are important to take in consideration when prioritizing between one item that needs to be refactored and implementing a new feature,
  - Competitive advantages
  - Specific customer values
  - Market attractiveness
  - Lead time
  - Maintenance cost

  - Customer long-term satisfaction
  - Risks
  - Penalties
- On a scale 1(not at all) - 5(very well), how aware are you about the term "technical debt?"

- Are you currently using any tools to monitor the quality of the software architecture today?
  - it yes, on a scale 1(not at all)-5(very much), how satisfied are you with that tool?

B.2 Questions after using the whiteboard(30 min)

- Based on your estimation, would you refactor this ATD item?(yes/no)

- Which factors did you concider?

- Would you have concidered more factors if you had more time? (yes/no)
  - if yes, make a list of those factors

- on a scale 1(not accurate at all)-5(very accurate), how accurate do you think the result of using the whitebaord was?

B.3 Questions after using the tool(30 min)

- Is the outputted suggestion from the tool different from your answer from using the whiteboard? (yes/no)

  If the output from the tool is different from the decision you took while using the whiteboard, does the tool change your mind? (yes/no)

- On a scale 1(not accurate at all)-5(very accurate), how accurate do you think the result of the tool was?

- On a scale 1(not useful at all)-5(very useful), are the factors taken in consideration in the tool any useful?
  - In hindsight, would you have have used them on the whiteboard? If no, why not?

- If you made a list of factors before, are any of these in the tool? (if yes, which one?)

- On a scale 1(absolutely nothing)-5(yes, very much), would you use the tool in your daily work?
  - If, 1, what would need to be changed so that it would be useful? (max 3 things)

- On a scale of 1 (none) - 5(all of them), Should the factors that were not considered in the tool, but on the whiteboard be included in the tool

- On a scale 1(bad)-5(excellent), how did the tool give feedback on the severity of the growth of cost of refactoring?

  If, 1, could you give an concrete example of something that could be changed?

- How would you use the results from the tool(ex discuss with architects/developers)

- On a scale 1(not at all)-5(very much), would the output help the communication with stakeholders?

- On a scale 1(not at all)-5(very much), would the output from the tool aid you in the planning process of "what to do next" in the development?
  - If 1, give a concrete example of how this could be done instead.

- What other features would you like from the tool?

C

Images of the Tool

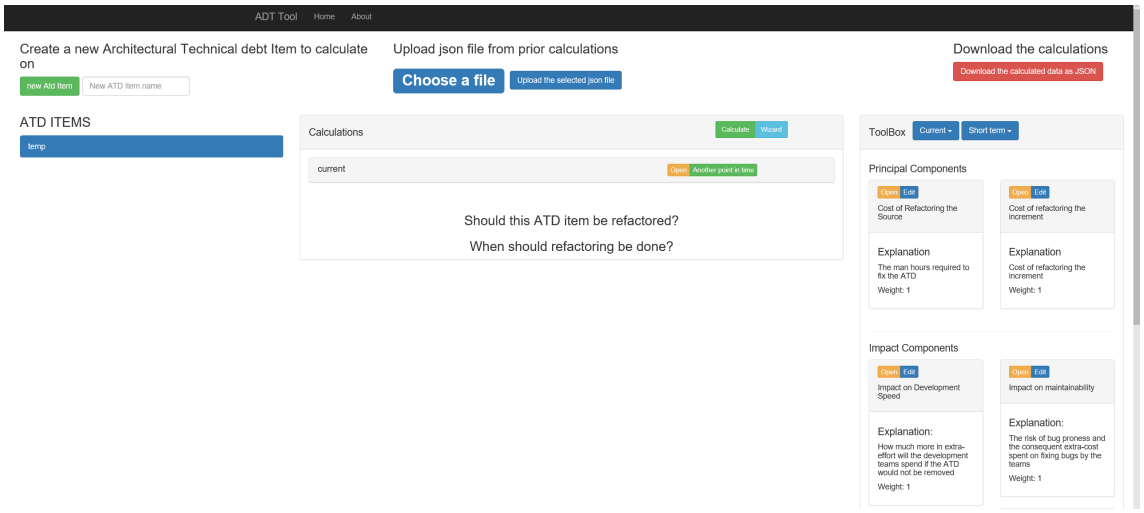


Figure C.1: The main page of the artifact

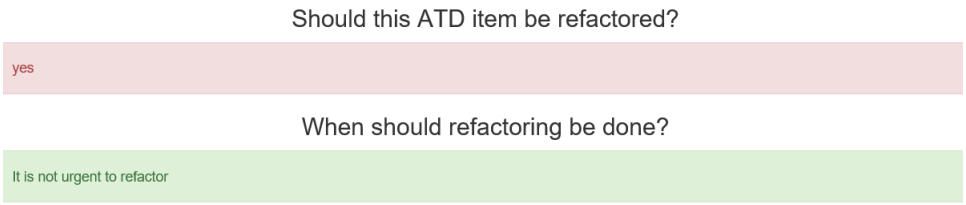


Figure C.2: The decision regarding if and when a refactoring should be done

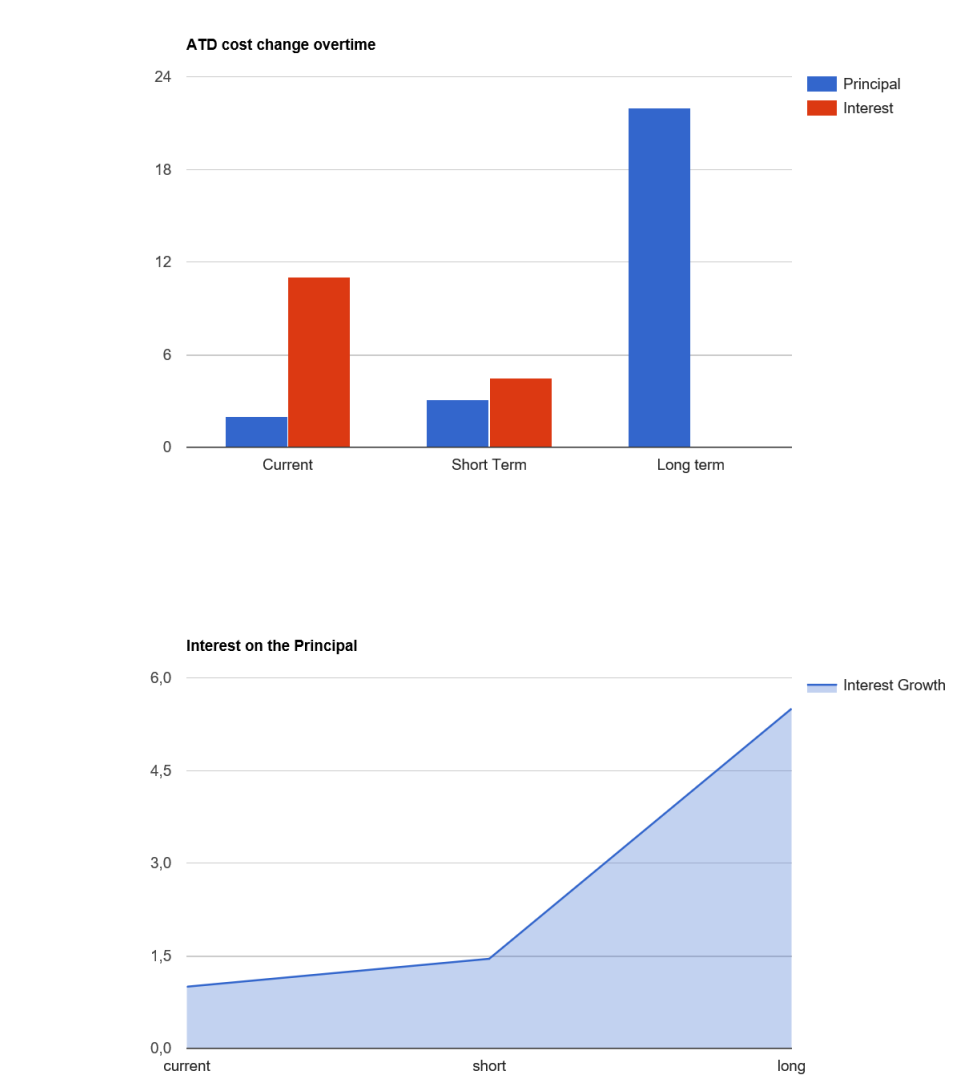


Figure C.3: Charts showing the growth of the cost of refactoring over time