Universiteit Leiden

ICT in Business and the Public Sector

When to update? A model to support non-functional software update decisions

Name: Jonathan Cammeraat
Student-no: s2671859

Date: 18/03/2022

1st supervisor: Prof.dr.ir. Joost Visser

2nd supervisor: Dr. W. Heijstek

Company supervisor: L. Blom

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden

Acknowledgements

I would like to extend my sincere thanks to Prof. Joost Visser and Leen Blom for their assistance at every stage of the research project. Their support and knowledge elevated the thesis in a way I could not have done alone. Also, I would like to thank Werner Heijstek for his support towards the end of the thesis. I would also like to thank my girlfriend and family for keeping me motivated and supporting me during this period. Lastly, I would like to say thanks to my colleague-students for their help.

Abstract

Keeping software up to date can prove to be a costly and time-consuming part of maintaining software. The involvement of product owners and any type of developers is almost inevitable in this part. The remainder of the required roles is dictated by the type of update. Software updates come in various forms and can be distinguished in numerous ways. One way to distinguish software updates is to look if the output of the update alters the behaviour of the product or if it does not. The updates that are within scope of this research are external library updates, refactorings, upgrades and language migrations. The updates that change software behaviour are not in the scope of this research. Also the updates that are essential e.g., (emergency) security patches, are taken out of scope as they cannot be deprioritized. The decision made at the moment of carrying out the update is currently based on past experiences and gut feeling.

This study aims to help both the developer and business by capturing decision criteria inside a model to achieve a more fact-based approach for deciding the moment of updating. Due to data-driven decisions businesses can achieve can achieve lower costs or incur less technical debt whilst updating. Developers will be able to evince that the non-behavioural updates need to be a higher priority. Task prioritization is based upon facts not gut feeling.

Investigation surrounding the decision-making of behaviour-preserving software updates is done within, a Dutch mid-sized software development and product. This investigation entailed interviews, ethnographic research and scenario- based interviews. A second set of interviews was used to obtain additional data on the language migration update. The ethnographic research was conducted by observing the decisions made during a Scrum sprint. Furthermore, a scenario for each update was drafted and tested amongst product owner. The goal of the scenarios was to see what factors played a role in deciding the moment to carry out an update. Validation of the produced model has been done on five aspects ease of use, compatibility, subjective norm, voluntariness and correctness.

The ethnographic research provided insight on the way of working within Centric. Conducted interviews presented various criteria on which the decisions were made. These criteria are the basis of the decision model that was created. By presenting possible scenarios concerning non-functional updates to product owners, we established that the decision relies on a large number of aspects and that it is situational. Part of the validation was to examine the participants opinion on the correctness of the model. The statement concerning the correctness did not receive universal agreement. However, the participants provided generally positive responses to the remainder of the statements presented.

The positive response on these statements can lead to at least two conclusions. The first one being that there is willingness to work with the model by the participants. The second one is that the model is perceived as useful. The less positive response on the correctness can be an indication of necessary improvements. An improvement on this topic could entail a model that fits the way of working better within Centric. However, this result can also indicate that further discussion or decisions need to be made beyond the model. While this study only included participants from a single organization, the developed decision model does not rely on Centric-specific characteristics. Therefore, we expect our work to be generalizable in large degree beyond the context of Centric.

List of abbreviations

TD –	Technical Debt
KQ –	Knowledge Questions
SIG –	Software Improvement Group
DS –	Design Science
FPA –	Function Point Analysis
SLOC –	Source Line of Code
PBI –	Product Backlog Item
API -	Application Programming Interface

List of figures

Contents

1. Introduction

This chapter provides information on the what and the why of the research. Also, the scope of the research is discussed in this chapter.

1.1 Background

In the last two decades a large number of software products have been developed. Internet technology innovation increases the capabilities of software causing a change in human needs [1]. This change in needs goes full circle after it functions as inspiration for more innovation.

Human needs are the biggest driver of software development. After identifying the needs, they get translated into ideas. These ideas are then, depending on the development method, developed to satisfy the concerned need.

After the deployment of software, the development process continues with product maintenance. Part of this maintenance is updating the software e.g., increasing the number of functionalities or reducing the number of faults [2]. Being up to date is a term that is widely used within software development companies. Taking on the quest of being up to date can prove to be difficult, it can be both costly and time-consuming.

In an IT-landscape there are many software products that are interwoven, this enlarges the possibility of a system being dependent on another [3]. More dependencies lead to a more complex system which also has impact on the parts that need to be looked at when updating. This correlates to the reason why updating can prove to be time-consuming, not only the concerning software needs to be looked after but also the dependencies of that software.

When updating software one of the dependencies can contain a breaking change [4]. This breaking change means that the software will not function as they should be once the update has been carried out. To ensure that the software functions as it should, action needs to be taken. This action can vary, most of the time it entails that the dependent software also needs an update. However, it can also be solved by replacing the dependent software by a similar product.

The term updating can be seen as quite broad. One way of splitting the term up is making use, where applicable, of the distinction that is used for requirements engineering. During the first phase of requirements engineering the requirements get elicited from the customer. These are divided into functional and non-functional. The functional requirements specify what the system should do, whilst non-functional updates specify how the system performs a certain function [5].

This research will focus on the updates that do not alter the behaviour of the system. These are mostly carried out to keep maintenance costs low but can also help keeping up with technical innovations. These updates generally are not driven by external pressure, as opposed to functional updates, resulting in deprioritization. This calls for good decision making in order to get the highest value out of the non-functional updates.

1.2 Relevance of research

Non-functional updates are, as mentioned previously, commonly deprioritized. This is due to a few reasons, one of them is the difficulty of estimating [6]. There are many aspects that can play a role when carrying out an update. A decision based on a gut-feeling of these aspects can be translated to uncertainty on the impact. The difficulty of estimating the output together with other tasks that are perceived to bring more value than non-functional updates results in deprioritization of them. Moreover, the likelihood of the decisions that do not result in the desired output increases due to the uncertainty. Which leads to people being less eager to make a decision, as they do not want to make a mistake. However, postponing the work is not always the right move.

In a report of CISQ [7] it is stated that badly engineered software is one of the primary causes of poor software quality. Non-functional updates have a direct impact on this as they tackle the non-behaviour aspects of the software. The software quality is connected to the maintenance of the product [8]. The maintenance part of the software development process is estimated to take up to 70% of the total cost of the entire process [9]. This shows the importance of non-functional updates that result in better-quality software and thus lower maintenance costs.

These updates can also play a role when it comes to security. Many software products make use of third-party libraries, also called library dependencies. These can save time and do not require the developers to reinvent the wheel [10]. However, the downside of these dependencies is that they can cause security issues. The libraries are maintained by a third party, meaning that the user is dependent on them. When the third party releases a new version, the user has to implement it into their structure themselves. This can go smoothly, but it can also be the cause of breaking changes. Developers do not always update their dependencies, because they either forget or have more urgent tasks to do. Libraries that are outdated are four times as likely to contain a security issue compared to up-to-date libraries [11]. This also indicates the necessity of non-functional updates.

Our research aims to increase the software developer's information provision on the decisions related to the moment of updating. This way the developers can make a substantiated decision based on the various related criteria. Aiming to optimize the trade-off between the costs and technical debt that are bound to the specific update.

An increase of the information provision on this subject can lead to a few things. The software developers can apply their resources on other aspects of their job, this way a more efficient usage of resources can be obtained. Next to that, fact-based decisions can be made concerning updates. Furthermore, it creates more control on the technical debt that resides within the company.

1.3 Scope

As mentioned there are a lot of different kinds of updates, and these can be categorized in different ways. This research focuses on updates that are based on two different distinctions. One of the distinctions made is on the effect of the update by looking if it impacts the behaviour of the product. The updates which change software behaviour are kept out of the scope of this research, because their priority is dependent on context and typically dictated by external factors. This mainly concerns the updates that add features or fix problems e.g., minor version patches [12] or bug fixes. Moreover, we make a distinction on the urgency of the update. Mandatory updates, for instance patches that fix security issues, are outside the scope. There is no reason to deprioritize this type of update, therefore there is no use in using an information model to make a data-driven decision on the prioritization of this update. The updates that are both behavioural-preserving and are not mandatory updates are referred to as non-functional updates in this research.

Keeping in mind the two requirements we deduced four updates. The four behaviour-preserving updates we will focus on in this research are updates that are refactorings, language migrations, language upgrades and library upgrades. There is no way to ensure that all the concerning updates do not contain any change in the behaviour. This is because updates do not get separated on this aspect which can lead to a mixture of behaviour-preserving and behaviour altering.

There are also two ways the non-functional updates can be triggered; these are proactive or reactive. The proactive updates require the developers to actively search for non-functional updates that can be carried out. The process of actively searching for these updates is taken outside the scope. The triggers in scope are the reactive ones, these are a reaction on an occurrence.

1.4 Technical research question and goals

In science research the exploration phase usually contains the drafting of research questions [13]. These questions provide a goal that gives direction to the research. This thesis will make use of the design science (DS) methodology [14], it will follow in particular the guidelines drafted by Wieringa [15]. In Design Science Research, research revolves around the design of an artifact. This research functions to determine an applicable design artifact for the concerning context. The designed artifact has the job satisfy the set goals.

The design science methodology makes use of goals. The goal structure of a design science project is shown in Figure 1. A distinction is made between the goals that are generated from the social context and the goals from the DS research. The arrows indicate which the goals derive from e.g., a prediction goal derives from a knowledge goal.

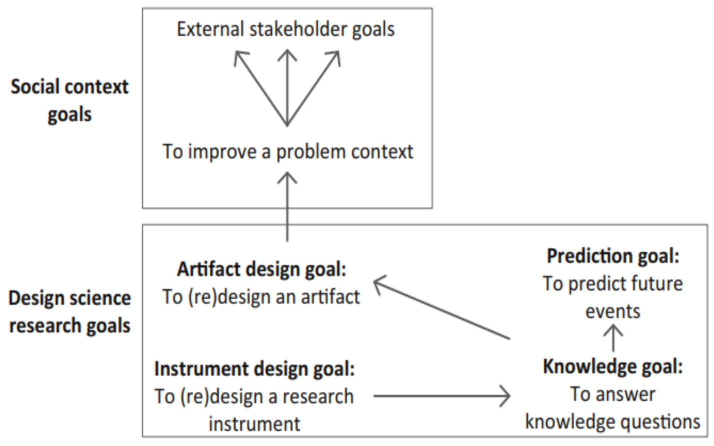

Social context goals
External stakeholder goals
To improve a problem context
Design science research goals
Artifact design goal:
To (re)design an artifact
Prediction goal:
To predict future events
Knowledge goal:
To answer knowledge questions
Instrument design goal:
To (re)design a research instrument

Figure 1 - Goal structure of a design science research project [15]

Traditional research questions can vary a lot. This is less of a problem with technical research questions. The fact that design science research will always have an artifact as output allows for a template to draft the technical research question. Below is a template shown that Wieringa [15] drafted for creating technical research questions.

Improve <*a problem context*>
by <*(re)designing an artifact*>
that satisfies <*some requirements*>
in order to <*help stakeholders achieve some goals*>

The chapter of Wieringa's book [15] regarding the relevance of the research covers the various reasons of why it is important to think about non-functional updates. This leads to the fact that a decision for updating non-functional updates can prove to be important. Basing the decision on gut feeling opens the possibility of making the wrong decision. This thesis aims to prevent those wrong decisions, leading to the following technical research question:

"**Improve** information provision concerning non-functional updates **by** designing a decision-making model **that satisfies** the need for more data-driven decision making **in order to** reduce future technical debt."

This technical research question derived from a research question we drafted earlier in the study. The design science methodology of Wieringa was the cause for this change. During the process of familiarizing with the problem and its context potential outcomes pointed to the creation of a design. This supports the usage of Wieringa's [15] methodology. Our initial research question is shown below.

"Which timeframe of carrying out non-functional updates provides the most business value when taking technical debt into consideration?"

1.5 Overview

The thesis is divided into the following chapters:

Chapter 1|	Covers the **introduction** of the thesis. This will provide a background together with the problem statement. Also, we mention the goals and knowledge questions in this chapter.
Chapter 2|	Provides information on the used **methodology** that will be used throughout the research.
Chapter 3|	The **literature review** provides an overview on relevant historical data that has been retrieved through prior research.
Chapter 4|	The **data gathering** aims to show how the various methods we use for data collection and what our findings are.
Chapter 5|	This chapter will provide the designed artifact: **A decision tool for non-functional updates**. The model is based on the information elicited during data gathering. The chapter also covers
Chapter 6|	This chapter will discuss the **acceptance of the decision model within the case company.**
Chapter 7|	The **discussion** will contain the various limitations and possible generalizations of our research.
Chapter 8|	The **Conclusion** provides answers to the drafted research questions together with the contribution this thesis delivers. Furthermore, it highlights topics that need further research.

2. Methodology

This chapter covers the methodology that is used in the research. We explain why we chose this methodology and how we apply it.

2.1 Design science

The desired output of our research is a model that supports data-driven decision-making when it comes to carrying out non-functional updates. The goal that has been set by the technical research question conform design science standards can only be achieved through qualitative research. The qualitative research methods we use to satisfy the goal are various types of interviews, literature review and ethnographic research. There are two widely used variants of design science, one written by Hevner [14] and the other by Wieringa [15]. These differ in the way that Wieringa separates design science into design and investigation, whereas Hevner only makes use of a design cycle [14]. Due to the lack of prior research a great part of the necessary data comes from investigation. Because of the increased focus on investigation, we decided to use the version of Wieringa.

The design science of Wieringa [15]. is, as mentioned above, divided into two parts which are investigation and design. The two processes work hand in hand with each other, as shown in Figure 2. The design part focuses on designing an artifact that improves the context in which the problem resides. The investigation provides knowledge and optionally new design problems to the design part.

However, the design science methodology does not only consist of the investigation and design. Figure 2 shows the framework of design science, it consists of three big building blocks with design science in the centre. These stones consist of the social context, design science and knowledge context. The first and latter provide the necessary context for the design science to be executed.

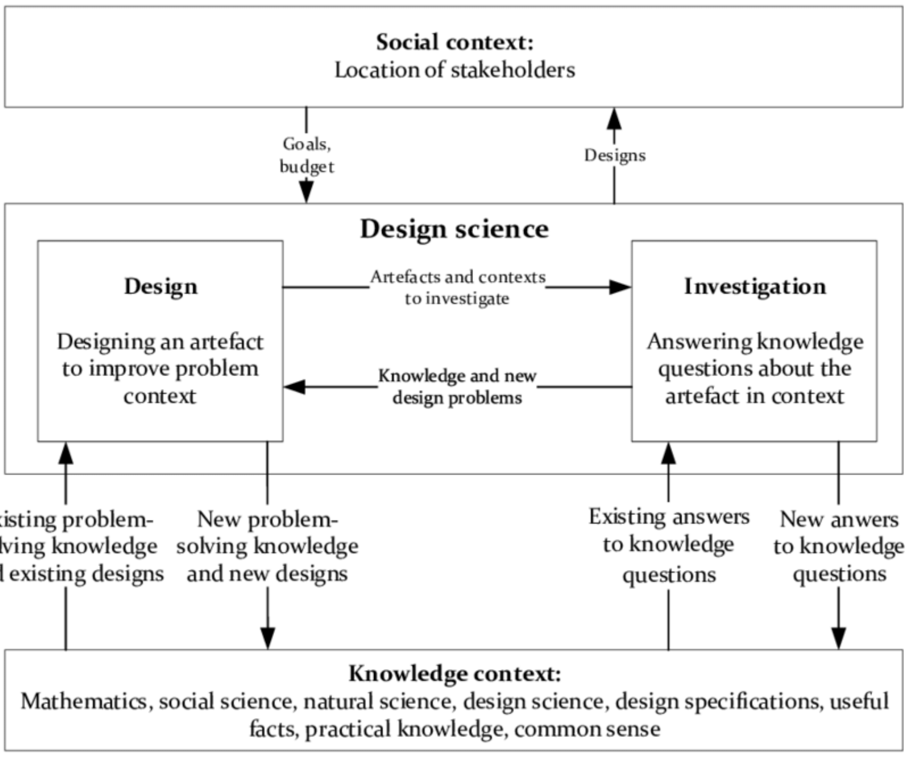
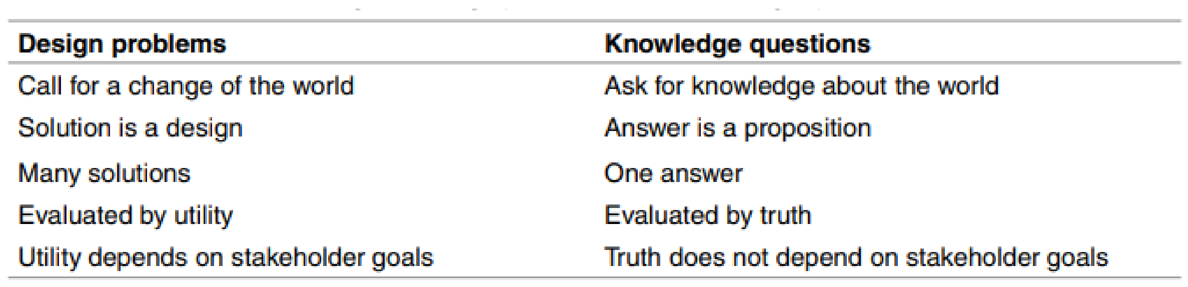

Social context:
Location of stakeholders
Goals, budget
Designs
Design science
Design
Designing an artefact to improve problem context
Artefacts and contexts to investigate
Investigation
Answering knowledge questions about the artefact in context
Knowledge and new design problems
Existing problem-solving knowledge and existing designs
New problem-solving knowledge and new designs
Existing answers to knowledge questions
New anwers to knowledge questions
Knowledge context:
Mathematics, social science, natural science, design science, design specifications, useful facts, practical knowledge, common sense

Figure 2 - Design science framework [15]

The social context provides a goal, this goal gets transformed into a design problem within the design cycle. Next to the design problems there are knowledge problems, these are created within the investigation part. The heuristics to distinguish the two are shown in figure 3. The knowledge questions create a structure way for acquiring the necessary knowledge to create the artifact. These knowledge questions are necessary to walk through the design cycle.

On the opposite side of the social context resides the knowledge context. This part functions as an input and an output for both the design and investigation of the design science. The potential input it supplies are the already existing information or design through prior research or other ventures. The output for the investigation can be newly obtained answers to knowledge questions and the design process supplies an artifact that improves the concerning problem context.



Design problems	Knowledge questions
Call for a change of the world	Ask for knowledge about the world
Solution is a design	Answer is a proposition
Many solutions	One answer
Evaluated by utility	Evaluated by truth
Utility depends on stakeholder goals	Truth does not depend on stakeholder goals

Figure 3 - Heuristics to distinguish design problems from knowledge questions [15]

When studying the design part of the design science more in depth there are three separated tasks connected. These three tasks are problem investigation, treatment design and treatment validation. The design cycle is a subset of the engineering cycle which also includes the implementation and evaluation of the artifact. The treatment that Wieringa refers to in two steps of the design cycle is the interaction between the problem context and the artifact [15].

The first step of the design cycle is the problem investigation this step contains the reasons for the project, referred to by Wieringa as goals of the project [15]. This section prepares the design of an artifact by learning more about the problem to be treated and is covered in the first chapter of the thesis. Next is the treatment design, this contains the requirements which are stated further in this chapter. Furthermore, the design of the artifact will be discussed in chapter 4 followed by the treatment validation in chapter 5. The treatment validation contains, as the name states, the validation of the treatment.

2.2 Requirements

The design task of design science has the desired output to deliver an artifact that improves a specific problem context. To come to such an output, various requirements need to be set. Requirements are a goal of the stakeholders desired properties of the artifact combined with the context assumptions.

Realizing well-drafted requirements is an important aspect of the design task. Obtaining these from stakeholders is a rare phenomenon according to Wieringa [15]. This makes it, in most cases, the task of the researcher to draft the requirements. A form of justification is necessary as the requirements do not originate from the stakeholder. This justification can be done in the form of a contribution argument, it dictates the contribution the requirement would deliver for a stakeholder [16]. The Contribution Argument (CA) is not always a sound argument, because it is a prediction.

For our research, we drafted the following requirements:

R1: The artifact can be used by roles on multiple levels within the organization e.g., on operational level by developers or on tactical level by product owners.

> **CA1**: If the artifact is in place, it should be able to be used by roles of multiple levels, because decisions on non-functional updates can be made on multiple levels within the organization.
> **CA2**: Making it understandable for multiple disciplines ensures ease of use, making this requirement also a non-functional requirement.

R2: The artifact should be able to be used for all the relevant non-functional updates

> **CA**: The stakeholder goal was to create more data-driven decisions concerning non-functional updates, this leads to the necessity of the artifact to handle all relevant updates. An example of a non-relevant non-functional update are patches, these are mandatory thus are not relevant for the artifact.

R3: The artifact provides a way for the users to make data-driven decisions.

> **CA**: The main goal of the research is to ensure more data-driven decision making; the artifact should support this.

R4: The artifact provides an overview of decision criteria.

> **CA**: The artifact shows the criteria that are necessary to form a decision on such matter. So that the stakeholder can collect the relevant data.

2.3 Knowledge questions

All the necessary information that should be gathered in order to get to a desired output are the knowledge goals. To reach these goals there should be knowledge questions drafted within the research. The answers to these questions form a proposition and there can only be one answer per question. These questions do not call for improvement but ask for knowledge about the world.

Wieringa [15] makes a distinction for these questions, it can either be an empirical knowledge question or it can be the opposite which is an analytical knowledge question. The empirical can then also be subdivided into descriptive and explanatory, as demonstrated in figure 4. The analytical questions are answered through conceptual analysis i.e., concepts need to be analysed in order to answer them. Whereas empirical questions require data collection and analysis of said data. All of the questions for our research are empirical. This means that none of the questions contain an analysis of a concept. However, we did analyse the concept ‘updates’ this is shown in 1.3.

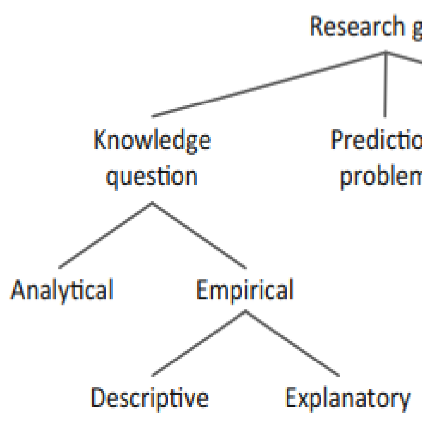


Figure 4 - A classification of research goals [15]

Shown in the figure are two subsets of empirical questions. In reality there are four classifications, two of each on a different level. Next to the option between descriptive and explanatory there can be a distinction made between open and closed questions. The closed questions include a hypothesis, whilst open questions do not have a specification of its possible outcome. The knowledge questions that are used in this research together with the goal of getting an answer for the concerning question are shown below.

KQ1: Do all non-functional updates have the same decision process?

Goal: The extent to which there is a difference between the updates can be cause for a different decision-making process.
Hypothesis: No, each process has update specific tasks which causes the updates decision processes to differ.

KQ2: What criteria are decisions of non-functional updates based upon?

Goal: Knowing on which aspects the decision resides on is an essential part of making the right decision

KQ3: Who is responsible for choosing the moment of updating?

Goal: The person who is responsible for this is going to be the user of the artifact making it an important piece of information to know. This can create targeted decisions when creating the artifact.

KQ4: What are the triggers for non-functional updates?

Goal: Knowing what starts the process of thinking about carrying out non-functional updates can cause a proactive approach for carrying out similar updates.

KQ5: Is there a most opportune moment to carry out non-functional updates?

Goal: The most opportune moment will provide the maximum business value.
Hypothesis: Yes, but not an exact moment as it relies on too many aspects.

3. Literature review

This section covers the information that we gathered from the knowledge context. This step allows for orientation on the current available knowledge. The required information that is not available within the knowledge context allows for more accurate self-gathered information.

3.1 Impact of design science research

A distinction can be made on the type of contribution the artifact would supply. This is based upon the maturity of the application domain and solution. A matrix of this has been created by Gregor and Hevner [17], this matrix is displayed in Figure 5. The two researchers state that three out of four of the quadrants are worth researching. The "routine" design is not suitable for design science research. Placing our research in one of the quadrants provides an argument of the usefulness of the artifact and the type of contribution it delivers.

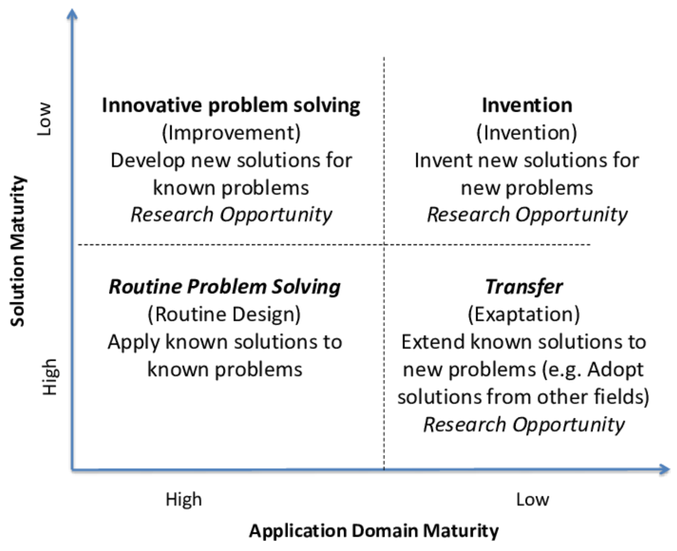

Solution Maturity
Low
High
Innovative problem solving
(Improvement)
Develop new solutions for known problems
Research Opportunity
Invention
(Invention)
Invent new solutions for new problems
Research Opportunity
Routine Problem Solving
(Routine Design)
Apply known solutions to known problems
Transfer
(Exaptation)
Extend known solutions to new problems (e.g. Adopt solutions from other fields)
Research Opportunity
High
Low
Application Domain Maturity

Figure 5 - Knowledge contribution framework [17]

The maturity of the solution is measured through the maturity of the artifacts that exist that can be used as a solution for the technical research question. This can range from high to low, with high signifying many artifacts that can provide a solution. The application domain is also measured using high and low, here the high refers to the maturity of the problem context.

The search started by looking at solutions for deciding regular updates to see if parts could prove useful. The search led to agile project management tools like Jira [18], these provided help by structuring the prioritization of the updates. However, they did not provide help on the prioritization itself. Also, artifacts that can be used for non-functional updates were not able to be found. Because of this the artifact resides in the low category of solution maturity.

The problem context on the other hand is quite mature since IT and its updates exist for years. The decision on which updates to prioritize higher than others based upon certain data has been around for the same amount of time. When combining the two maturity levels we can conclude that the artifact is an innovative problem-solving artifact. This means that the research is not a routine problem solving and is able to provide a contribution.

3.2 Related work

Three of the main knowledge categories that were studied were technical debt, non-functional software updates and decision making. The research is about improving the decision making on these updates to reduce the amount of gained technical debt. This makes the three mentioned topics important aspects to have knowledge about. The keywords we use in effort of finding the information on Google Scholar are stated below.

Keywords: Technical debt, decision making, dependencies, technical updates

Technical debt

The term technical debt was introduced in the nineties by Ward Cunningham. The first use of the metaphor was Cunningham's way of providing argument to a refactoring [19]. Over the years the term developed and is now used for anything that implies lower immediate costs at the expense of higher long-term costs. This way there is a measure for the consequences of forcing deadlines.

Incurring technical debt can happen in two ways. According to McConnell [20] this is unwillingly or willingly, which he refers to as unintentional and intentional. The intentional technical debt can be incurred by intentionally making shortcuts in order to meet deadlines. The unintentional technical debt can be incurred due to bad decisions that lead to rework in a later stadium, such as bad coding.

Technical debt is a container definition that can be divided into various dimensions. The dimensions show the dimension of the software in which the technical debt incurred. Furthermore, there are distinctions between attributes, causes, and effects. These categories and their distinctions are described by E. Tom et al., in their paper 'An exploration of technical debt'. An overview of their results is shown in table 6 [21].

Dimensions	**Attributes**	**Cause**	**Effect**
Code debt	Monetary costs	Pragmatism	Impact on morale
Design and architectural debt	Amnesty	Processes	Impact on quality
Environmental debt	Bankruptcy	Prioritization	Impact on productivity
Knowledge distribution and documentation debt	Interest and principal	Attitudes	Impact on risk
Testing debt	Leverage	Ignorance and oversight	
	Repayment and withdrawal		

Table 1 - Summary of results from 'an exploration of technical debt' [21]

The non-functional updates can play two separate roles when it comes to technical debt. It can either incur more technical debt in one of the two ways described earlier. An example of incurring more debt is when an upgrade leads to shortcuts or shortcomings in the design and architectural dimension [21].

However, as has been stated it also plays a role in reducing the amount of technical debt. B. Curtis found, in his search for a quantifying method of estimating the TD within a business application, that each line of code that has to be changed to repair quality issues equals 3 dollars and 61 cents on average [22]. This amount is translated into interest on top of the current maintenance costs. In figure 7 there is a graph shown of the costs against the time of maintenance costs. The more time passes the more interest is being racked up on top of the already technical debt. Resulting in an increase of the initial amount of 3 dollars and 61 cents. Next to that, B. Curtis found that the average varies between each programming language. The highest cost per line of code being Java-EE with 5 dollars and 42 cents [22]. The earlier this is dealt with the less interest that is being accrued. This relates to KQ5, which is concerning the moment of updating. When deciding to prioritize the non-functional update to a later moment the interest could rise and thus the costs of the business rise.

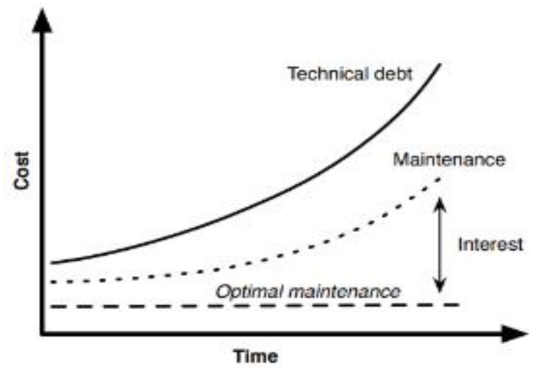


Figure 6 - Technical debt and its interest grows over time if not resolved [23]

Non-functional updates can help reduce the amount of technical debt. This is especially the case with refactorings, these can be used to target technical debt removal [24]. The removal of technical debt is most commonly done by the person who introduced it into the company. Making the existence of technical debt a possible trigger, this can be used as a part of the answer of KQ4. Having knowledge of the amount of TD the company has accrued over time can support the argument to carry out a non-functional update, if this impacts the TD in a positive manner. However, this is only when the update proves to create an impact on reducing the technical debt. Making effort to incur technical debt willingly opposed to unwillingly can prove helpful when keeping track of the current amount of debt.

Software updates

There has been extensive amount of research into the knowledge context of updates. The necessity for updates is triggered by the need to evolve software. When diving deeper into the reasoning for the constant need to evolve, the question 'why?' can be asked. The answer to this leads to the ability to compete in today's economy [25]. This constant evolution goes hand in hand with the changing needs of customers.

A way to evolve software can be done by carrying out updates. These updates can come in any form; it could be adding a feature, but it could also mean a refactoring of bad quality code that reduces technical debt which leads to lower costs. In 1974 M. Lehman et al. formulated eight laws of software evolution. The second law states that software gets increasingly more complex when it evolves and no action against it is taken [26]. This law can be used as an argument in answering KQ5, because the effort of the update possibly increases. When it is chosen to delay the update or prioritize it low, the software can evolve through functional updates and thus increases complexity. This complexity relates to the code becoming messier and harder to apply further changes [25]. Next to the changes being harder to apply, the amount of refactoring that is necessary grows too. The assumption would be made that companies would invest time into limiting the growth in complexity. However, according to Visser [25] this is generally not the case. He mentions that refactoring and commoditization are easily forgotten and prioritized low due to them not having external pressure. This provides support for the argument that waiting with certain updates increases the effort.

There were not many scientific papers concerning non-functional updates. However, for the library dependencies there is information available. According to a study on whether developers update their libraries or not it is found that 81,5% of the studied systems contain an outdated dependency [27].

This is quite a large number; this number can be used as one of the triggers that are asked within KQ4. The study also mentions that the same developers that experience vulnerable dependencies do not act on security advisories. Also, 69% are unaware of said vulnerable dependencies even though the systems rely heavily on the dependencies. Just like the refactoring the library dependencies are not likely to be prioritized high. Furthermore, dependencies that are outdated are four times more likely to have security issues [11]. There is also information to be found on the way of measuring how outdated the dependency libraries are. A. Zerioua li et al. introduced the term technical lag in their paper [28]. Technical lag refers to the delay between the most recent version and the currently used version. Another definition that is used for a similar reason is dependency freshness [11]. This term looks at the desired dependency whilst technical lag looks at the newest version. This is, probably in most cases, the same. A way of measuring these two terms can be done by using Libyear. It is a single number that represents the number of years between the current version and latest version of the library in question [29].

When deciding when the best moment is to update as mentioned in KQ5, all the aspects that are relevant should be taken into consideration. In an article written by D. Parmar relevant aspects of upgrading software are mentioned [30]. The aspects that are mentioned are necessary employee training, lost productivity, and operational considerations.

Decision making

The topic of decision making is also taken into the literature review. The reason for searching information on this topic was to function as support during the interviews. The potential information could provide us with insight on the reasoning behind made decisions this can help us understand the situation better. This possibly leads to more specific questions or follow-up questions.

Decision making is a process of making a choice from a number of alternatives to achieve a desired result [31]. There are two basic models to be acknowledged, these are the bounded rationality model and rational model [32]. The latter includes six steps, these steps can be iterated if necessary, which are shown in figure 7. This model is based upon the assumption that the decision maker has all the necessary information. With the bounded rationality model this is not the case, this type of decision making is based on incomplete data. This type of decision making is used concerning updates because not all the information concerning the process is available.

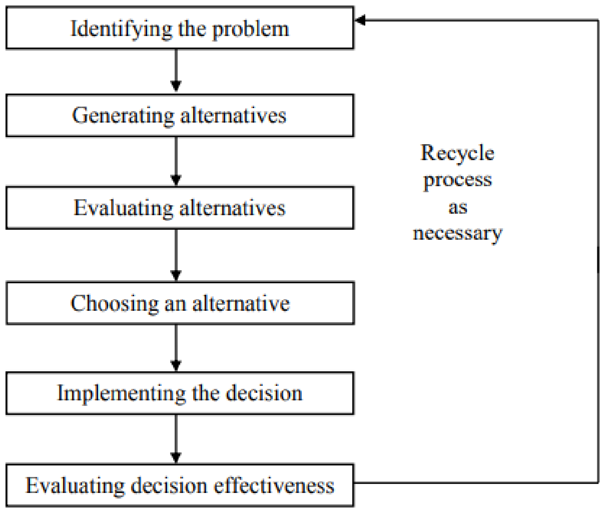


Figure 7 - Decision making process in the rational model [32]

4. Data gathering

The approach together with its output that we used for the data collection is discussed in this chapter. Our main instrument for data gathering was to conduct interviews and ethnographic research by observing development activities. We gather the data within Centric, this is a company that offers a variety of IT-related services to other businesses. It is a Dutch company that supports locally and abroad, the company has offices in nine different European countries. Centric is mostly active in the public sector, they develop their own products and deliver services with them.

A few of the interviews are on the shorter side. The shown length is based on the time of the recorded part. Most of the recorded parts did not include small-talk, this was done before the recording started. Next to that, alongside the invitation we provided the direction and purpose of the interview. This made it so that briefly discussing it was sufficient. With this in mind the shown length consists mostly of effective time talked on the subject and provided an in-depth view on the subject.

	#	Role	Interview focus	Date	length
First set	A	Technical lead	Privacy Workspace	08-04-21’	37 min
	B	Product owner	Key2Begraven	13-04-21’	41 min
	C	Frontend developer	Key2Belastingen	04-05-21’	46 min
	D	Frontend developer	Styleguide	05-05-21’	50 min
	E	PL/SQL developer	Key2Financien	11-05-21’	31 min
	F	Tech lead	Key2Burgerzaken	18-05-21’	31 min
Second set	G	Strategic product manager	Yellowstone	17-06-21’	37 min
	H	Program manager / Product manager	Nieuwe Maas	07-07-21’	50 min
	I	Line manager	Tiber	09-07-21’	41 min
Scenarios	J	Product owner	Scenario 1, 4	20-07-21’	-
	K	Product owner	Scenario 1,2,4	25-07-21’	-
	L	Developer	Scenario 2	27-07-21’	-
	M	Product owner	Scenario 1, 3	28-07-21’	-
	N	Product owner	Scenario 3	02-08-21’	-

Table 2 - Interview references

4.1 Interviews

This part will cover the method and results of the interviews.

4.1.1 First set of interviews

Before we could gather any data using a method, we had to do preparations. A part of the preparations was the literature review to analyse the problem. The orientation enabled us to draft an interview protocol, as shown in appendix A. This protocol includes generic questions which are suited for every interview, but it also includes specific questions. These questions were concern the product the interviewees were working on or the team they reside in. A total of six interviews are done in the first set of interviews, in a semi-structured way. This type of interviewing provided structure combined with the possibility of follow-up questions. These follow-up questions allowed the interview to go into more depth.

The chosen interviewees were based upon internal data from Centric. The internal data was retrieved through the applications SonarQube and Sigrid. SonarQube is an open-source platform developed by the company SonarSource. The application keeps track of the quality of the projects, it does this by showing results on maintainability, security, and reliability. Centric uses the software to keep track of code smells, bugs, and vulnerabilities. Sigrid is developed by the Software Improvement Group (SIG), it focuses on the maintainability and security vulnerabilities of systems. The software provides a calculation of the maintainability based on module coupling and unit complexity amongst other things. This calculation can score ranging from 1 to 5, Centric uses this calculation as a way to justify projects or to draft goals. Moreover, the software from the SIG displays the vulnerability risks of systems and libraries.

We chose the first two interviews based on potential security breaches. These were shown on the software Sigrid, the software showed that there were two systems with a vulnerability risk. The next two participants were chosen based upon a high TD score on SonarQube. The last two interviewees are chosen based upon a project they are working on. This project concerns a language migration, the made decisions are still fresh in mind thus making them a good source of information to interview.

The reasoning for conducting the interviews was to get a good view of the problem context from the stakeholders which reside in the context. In order to get to the goal of a clear problem context the interviews functioned as exploratory. This resulted in questions asked about the way of working and current decision making. Furthermore, the two topics technical debt and dependencies were tackled as well.

Each interview included a portion of interviewee-specific questions, the remaining questions kept the same each interview with minor changes. The products which contained potential breaches received specific questions e.g., if these were known and the possible origin. Also, unclear questions were altered if they did not produce the desired results. Furthermore, questions were added if they could increase the information provision.

4.1.2 Content analysis first set

All the interviews were recorded with consent. This made it possible to transcribe all the interviews. The software product Express Scribe was used, this software enhanced the speed of the transcribing process. The software provided hotkeys i.e., rewind in time, alteration of speed and start or stop. This proved useful for the interviews that were done in Dutch, it allowed for easier translation.

The content analysis of the transcribed interviews is done by an approach described by E. Blair [33]. This approach is a combination of the two methods open coding and template coding. The latter is a more structured method and makes use of prior research whereas the former does not and is a first step for a grounded theory.

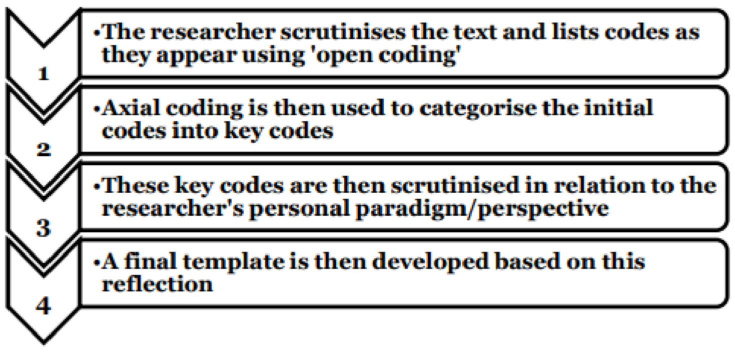

1
•The researcher scrutinises the text and lists codes as they appear using 'open coding'
2
•Axial coding is then used to categorise the initial codes into key codes
3
•These key codes are then scrutinised in relation to the researcher's personal paradigm/perspective
4
•A final template is then developed based on this reflection

Figure 8 - Combined approach used to develop the data coding template [33]

Figure 9 shows the four steps concerning the mixed approach. The tool that we use to execute this approach is MAXQDA. This is a tool used to analyse data for qualitative and mixed methods research [34]. The goal of the content analysis of this part of the data gathering was to acquire findings that would lead to an answer on the technical research question. The initial thought of executing this was to find theories. The data gathering of our research does not contain confirmatory methods, this makes it difficult construction a theory. However, we derived to a basic theory concerning criteria on which decisions are made for non-functional updates. The theory starts with that not every decision is made in a rational manner. Part of the whole decision-making of non-functional updates are based upon gut feeling. Even so, the decisions that are made by the employees entail a level of consensus on the important criteria for the concerning update, assuming a rational process. The person responsible for making the decisions is dependent on the type of non-functional update.

The criteria that were found during the first set of interviews are shown in Table 4, these help with answering KQ2. These criteria are used in the artifact, the source of these is shown in the last column. Next to the findings and criteria in the first set of interviews, we noticed a distinction. This distinction was on the topic of the type of update and created support for answering KQ1. During the interviews general questions were asked about non-functional updates. It has become clear that there is a large difference between the type of updates that do not alter the behaviour of software. The attributes in which the type of updates can differ are desired output, role that decides the moment and the effort. The type of updates for which a distinction is made are language migration, upgrade, external library and refactoring.

As mentioned, we noticed that there are different roles that are responsible for deciding the moment. In management there are three levels of control to be acknowledged, operational, tactical, and strategical. Each of these levels of control included one of the roles that were responsible for the decision. In the case company one of the roles on strategic level is the product manager, for tactical level there are the product owners and operational level the developers.

In order to obtain information concerning KQ3, a direct question was drafted concerning the deciding party. The upgrade is decided by the product owner together with the development team of the concerned product. This is in most cases also done with the smaller updates like library dependencies and refactoring. However, the moment of carrying out one of the last two can also be done by developers if these do not require a lot of work. The larger updates like the language migrations are decided on strategic level. These findings provided an answer on KQ3.

The first interviewees were chosen upon vulnerabilities shown on the maintenance software from Sigrid. After questioning the developers working on the product about the existence of these vulnerabilities it became clear that they had no knowledge of it. When zooming in on the place where the vulnerability showed up, it showed that there was a risk concerning password recovery. However, the developer explained that the product makes use of a different password recovery and that it was not really a vulnerability because of that. The interviewee was already known with this fact and he said that there was no need to make a decision for updating. This shows a measurement error, this calls for logically investigating to see if it is an actual problem before making any decisions.

The first set of interviews made clear that a distinction could be made for the type of non-functional updates. This distinction also became clear on the level of the roles that decide when to carry out the update. The roles that were interviewed in the first set included developer and product owner. These two roles cover all the updates except the language migration update. This led to the lack of insight of the product managers, which are responsible for language migration updates.

4.1.3 Second set of interviews

Similar to the first set of interviews the second set uses a semi-structured method to conduct the interviews. However, the interview protocol was slightly different from the first set. The second set included a distinction between exploratory and confirmatory questions. The exploratory questions are introduced to retrieve missing data on the topic of language migration. This data was not able to be retrieved from the first set of interviews. The confirmatory questions were drafted based on the project approval request (PAR) documents to confirm theories. The protocol of these interviews is shown in the appendix.

Interviewees were selected on their level of involvement in language migration projects. Their role in these projects had to be one with involvement in the end decisions. These roles know the most about the project and the reasoning behind the decisions. After examining the PAR documents of the project the interviewees were chosen. We have interviewed four people in three interviews during the second set of interviews.

Results of this part of the data gathering showed us that there are more criteria concerning non-functional updates. Giving us a broader answer to KQ2, as addition on the already elicited criteria from the first set of interviews. Moreover, it provided insight on the steps that are necessary to be taken before carrying out a language migration update. This provided a partial answer to KQ1, since it showed that there are different or additional steps concerning the various updates. In addition, it allowed the possibility to research the trigger for each of the projects. Resulting in being able to provide an answer on KQ4 for the language migration part, the results are discussed in chapter 4.3.

4.2 Ethnographic research

Whilst conducting the interviews, a project came to light which was a point of interest. The aim of the project is to renew the current UI technology. This renewal can be seen as a non-functional update because it does not affect functional behaviour. There were two ways to obtain the information from this project. The first option was to conduct interviews and the second option was through ethnographic research.

After taking multiple factors into consideration, we decided to use ethnographic research. To get a close look into the project and to support the data collection from a different perspective. Next to that, the ethnographic research provided a way to validate the answers that were given on the way of working that were asked during the first set of interviews. This way it could be validated to see if different projects used the same methods and if they were carried out as told. Furthermore, it was always possible to interview one of the stakeholders of the project during the ethnographic research or afterwards.

The execution of this approach was through observing a scrum sprint of one of the development teams. By observing a sprint, insight has been gained on the interaction between the participants of the sprint and the decision making during a project. It also allowed us to see which of the attendees had the most information and were, looking from a research perspective, the most interesting to interview.

By attending the sprint, we became acquainted with the way the team worked on the product. This led to seeing how the planning of the product backlog items (PBI’s) was done. This started in the sprint planning, which is the event that kicks off the sprint. In this event the development team and the product owner go over the still unresolved PBI’s of the previous sprint. Next to that, the team tackles the PBI’s that they want to be done in the upcoming sprint. Each PBI is equipped with a number for its effort. This number is chosen by the team based on previous experiences and gut feeling. This numbers represent the number of hours necessary to complete the task. In order for the team to know what PBI’s will be taken on in the sprint backlog the number of available hours is calculated for that specific sprint.

The effort of the leftover PBI’s of the previous sprint are, in most cases, reconsidered and placed on the sprint backlog first. The remainder of the hours are filled with new PBI’s. The third knowledge question concerning the person responsible for deciding the moment of updating can be partially answered when looking at who is responsible for prioritizing the PBI’s. This is because the non-functional updates are also formed into PBI’s to conform to the Scrum methodology and prioritizing these is a form of deciding when they are carried out. The ethnographic research shows that more than one role is responsible for the prioritization. It is a joint decision of the development team and product owner. However, the product owner has the last word on this decision, making the product owner the end responsible.

4.3 BPMN 2.0 process model

The separation of the various non-functional updates and the first knowledge question resulted in the need for an overview of the differences between the updates. Also, an overview could be used to see if there is any information still missing concerning the processes. We decided to use a flowchart diagram to create an overview. The flowchart method used is BPMN 2.0, the values of this method are predefined which makes it easy to use and clear for every party involved.

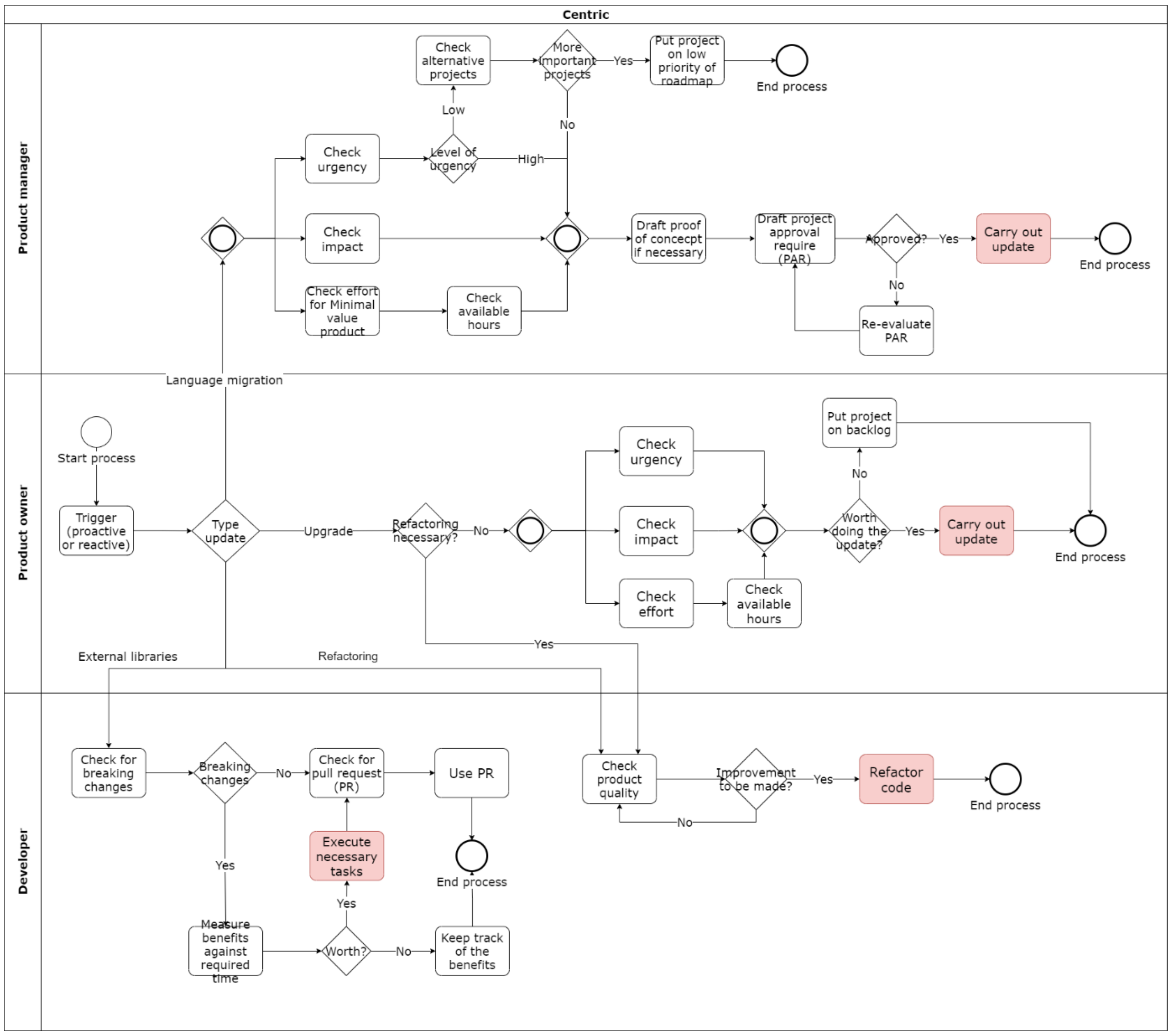


Figure 9 - BPMN 2.0 non-functional updates processes

Figure 10 shows the result of the process model. The pool is named after the case company, this is because the processes can be different in other companies. Furthermore, there are three different lanes, these represent the roles connected to the updates. As mentioned before these three are developer, product owner and product manager. The figure shows that the developer is responsible for two types of updates whilst the product owner is only responsible for one. This is not entirely true because they are discussed and prioritized within the scrum team in which both roles reside. However, refactoring and library dependencies can be updated by the developer alone when the work to carry it out is deemed small enough. This led to the separation and placement of refactoring and library dependencies.

The beginning of the processes starts with a trigger, this trigger can either be reactive or proactive. The proactive triggers are the ones which the employee actively seeks e.g., a developer searches through GitHub or another versioning control software to see if there are newer versions. On the other end are the reactive triggers, which can be when looking at the same type of example, be a notification received through the mentioned software. During the research various triggers came to light. The triggers help answering knowledge question 4, we came across the following triggers shown in table 2.

Type of update	Trigger	Type of trigger*
Language migration	The **stop of support** on an externally hosted software in use	Reactive
	A **newly acquired development team** can shift the language knowledge within a company, resulting in lack of expertise of the programming language in use	Reactive
	The application has been deemed outdated making it a **legacy system**	Both
Upgrading	A system has found a **security vulnerability** that needs to be solved	Reactive
	The maintenance software in use, **SIG** within Centric, shows a **score** which is too low **	Reactive
	In order for other updates to be carried out **dependent systems** could need an update to prevent a breaking change	Reactive
	The want or need to reduce the amount of **technical debt** to reduce. It can be reactive when the amount of technical debt is shown in a system, and it is higher than the guidelines prescribe. It can be proactive when the software quality is perceived to be too low.	Both
	Removing the chance of having any trouble with the system by always being up to date and thus removing the **technical barrier**	Proactive
External library	A system has found a **security vulnerability** that needs to be solved	Reactive
	In order for other updates to be carried out **dependent systems** could need an update to prevent a breaking change	Reactive
	The maintenance software in use, **SIG** within Centric, shows a **score** which is too low. **	Both
Refactoring	It can occur that the software quality or framework change is a **requirement** for another update	Reactive
	Drafted **code guidelines** can prove to differ in a negative way.	Reactive
	Reducing **technical debt** by making non-behavioural changes to the system to reduce maintenance costs	Proactive
	The maintenance software in use, **SIG** within Centric, shows a **score** which is too low, refactoring can be used to lower it**	Both

Table 3 - Various triggers of non-functional updates process

* This can be three options: reactive, proactive or both. It can be either option when the type of trigger is both.

** The score being seen as 'too low' can be due to set guidelines or goals.

Moving further into the process the process splits into the different types of updates. At the end of the processes, except library dependencies, there is a task highlighted with a red frame. These tasks caused for a question to be asked: How is the actual moment of carrying out chosen? This question also relates to KQ5, making it more interesting to answer this. The absence of this information resulted in scenarios to be drafted. These scenarios have the goal to answer the question.

4.4 Scenarios

The creation of the process model revealed that more information was required to be gathered. This part of information supports the process model together with answering the last knowledge question. The information was not retrieved through the interviews and ethnographic research, this required a different approach. The method that we chose to use to retrieve the information was conducting scenario-based interviews. The creation of scenarios allows for presenting these as an example to the participants. When presenting the scenario we also asked one or more questions to see how the person would react in the given scenario. The roles that are used for this method of gathering data are the same as used in the interviews. For every type of update there is a scenario drafted, these are shown below:

Scenario 1 - upgrading*:*

The front-end software language requires an update. The language will upgrade from version 6.0 to version 7.0. The backlog items are clear, the number of necessary hours will take up a significant number of hours.

- How is decided when to take on these backlog items?

Scenario 2 - refactoring:

A goal is to keep the software product above two stars of maintainability on the SIG scale. Recently the product went below the two stars. This triggers to look for improvements, after researching it turns out refactoring is the best way to increase the maintainability. All the backlog items are known, an example of these backlog items is to improve the module coupling.

- What criteria do you prioritize these backlog items upon?
- What makes you decide the moment to work on these backlog items?

Scenario 3 – Language migration:

Within the company there is an own developed management information system that uses programming language A. It has been in use for at least 20 years. The supplier of the software language notifies that support for the language will end in three years. The company has the resources to make use of the system without having technical problems even after the three years. However, combined with the opinion of the customer that the system has an outdated look and feel a change is required. The change involves the adoption of a different software language which can provide a more modern look and feel.

- What factors would you take into consideration when updating this?
- When do you decide to start language migration?

Scenario 4 – Updating external libraries:

You make use of several external libraries. It has come to light that two of the libraries need an update. One of the libraries does not contain a breaking change, the other does.

- How do you decide when to carry out the updates?
- Does the breaking change make a difference in approach?

It was evident that there were no straightforward answers to the questions asked after presenting the scenarios, the answers were very divers. The answers received included that there were many factors that had to be answered first. The mentioned factors matched with the decision criteria that have been elicited through the interviews. Furthermore, the answers included that the decisions were made based upon three different categories: effort, impact, and urgency. One of the respondents mentioned that it was dependent on the "breathing space" they had. This referred to the urgency of the update compared to the urgency of other pending tasks. Another respondent said that it was dependent on the available capacity. This showed that having knowledge of the effort could prove useful, because that would indicate how much available capacity is necessary. Other responses put the impact of the update as the key information of prioritizing the update. It was rationalized that if the impact was known some sort of urgency could be connected to the update, making it easier to prioritize. The responses made it clear that specifically the urgency and effort were important. With these findings it can be concluded that it is imperative that information about one or more of the mentioned categories has to be known before a decision can be made. This shows that there is a need for a model that displays the necessary information.

5. A decision model for non-functional updates

This chapter provides the outcome of the investigation in the form of a design. The aspects concerning the moment of carrying out the updates are also discussed.

5.1 Decision-model design

The desired output of this study is an model that solves the described problem context. This is in line with the used methodology design science. Having this is mind we thought of ways to give form to the model whilst gathering data. It started off by looking which data would be necessary to incorporate into the model. Preliminary thoughts came to making a distinction between the size of the updates. However, this idea soon got disposed of as there was a difficulty creating a point of measurement which would enable for a distinction to be made. Also, even if this was possible there was not a clear gain of diverging the updates in this manner. Further on we looked at making a model which would be able to create a decision on the moment of carrying out the update. This involved drafting an equation that would provide results on the different outcomes of different timeframes. The equation we derived on was the following.

$$Update\ moment = \frac{FTE+lp}{FTE+o}$$

The topside of the equation represents if the update would instantly be carried out. It takes into consideration the necessary hours (FTE) and the lost productivity (lp). The bottom side represents a moment in the future where also the necessary hours (FTE) are taken into consideration. This could be different from the timeframe on the topside as the complexity of the system can change and play a role. Moreover a difference could be if there is a lack of resources and externals have to be hired in order to reach set goal. Furthermore, the overspend(o) on the incurred technical debt is taken into consideration. The outcome of the equation would tell which timeframe has the highest business value. Anything above 1 would suggest that the cost of the earlier timeframe is higher, making the other timeframe more cost efficient. Unfortunately we were not able to pull through with this model as it was too difficult retrieving the necessary data. Also, the model was not able to be made universal for the different type of updates. This could be seen as events such as trends could interfere with the model when it concerned language migration updates. During our study we came across this. Two similar project were running within Centric, one had more resources put into the project. This project finished earlier, months after the completion the organization decided to make use of SaaS solutions. This also concerned the two projects, the project that was not done yet had an easier way of adapting resulting in less costs made than the already finished project which had to start over.

The design previous to the final design was also focused on being able to create a decision. This model takes the three main categories urgency, impact and effort as points of input. This version would requires the user to what extend the categories are (low, medium, high) concerning the update as input. Then based off the input the model would form a direction on which the decision should go. However, as this would require either a lot of different pre-drafted directions as there are many different outcomes. Thinning out the number of possible outcomes could result in a model that does not provide value to the user. Next to that, a way to measure all three of the categories can take up unnecessary time and could also prove to be a difficult task. With these cons we decided to approach it from a different perspective. The model we derived to is shown in Figure 10. It does not try to push the user into a decision but rather it would help the user make a decision on its own. This ensures a certain flexibility as the people within the deciding party can combine their experience with the model. Furthermore, unique situations can be tackled more easily. The final design also makes use of the three extracted categories.

This model provides an approach to increase data-driven decision making of non-functional updates. The model supports the decision process of all the previously mentioned updates. There are two main sections to be noticed within the model. The left side consists of three categories which are subdivided into criteria. These criteria have been acquired through the used data gathering methods. The right side consists of the researched updates. The user of the model must provide a type of update as input. Once the type of update is clear the user must decide which categories, if not all, are necessary to make a decision on when to carry out the update. This can be done by answering the stated questions of the relevant criteria. The criteria are perceived as relevant when the question is inside a column of the concerning update. Additional explanation on the criteria and type of updates can be found on the pages below.

		Type of update			
*	**Criteria**	*Refactoring*	*Language migration*	*Upgrade*	*External library*
Effort	Change size	What is the amount of LOC that has to be altered?	What is the amount of LOC that has to be altered?	What is the amount of LOC that has to be altered?	
	Breaking changes			What is the extra effort to get rid of the breaking change? (Backwards compatible?)	What is the extra effort to get rid of the breaking change? (Backwards compatible?)
	Dependencies		What are all the dependent factors?	What are all the dependent factors?	What are all the dependent factors?
Urgency	Project support	Is the update worth the time and effort looking at the end date of the project?	Is the update worth the time and effort looking at the end date of the project?	Is the update worth the time and effort looking at the end date of the project?	Is the update worth the time and effort looking at the end date of the project?
	External support		Is there an end date on the support of the programming language?		Is there a notification from the external supplier on stopping support?
	Future innovations		Is there an innovation which could affect the change in any way?		
	Market		Is the market in which the product resides highly competitive?	Is the market in which the product resides highly competitive?	
	Product lifecycle	Which phase of the product lifecycle is the product currently in?	Which phase of the product lifecycle is the product currently in?	Which phase of the product lifecycle is the product currently in?	Which phase of the product lifecycle is the product currently in?
	Current Technical debt	How much technical debt has the product currently incurred?	How much technical debt has the product currently incurred?	How much technical debt has the product currently incurred?	How much technical debt has the product currently incurred?
Impact	Potential Technical debt	What will the effect of the update be on the amount of technical debt?	What will the effect of the update be on the amount of technical debt?	What will the effect of the update be on the amount of technical debt?	What will the effect of the update be on the amount of technical debt?
	Security		Does the update affect security in either a positive or possibly a negative way?	Does the update affect security in either a positive or possibly a negative way?	Does the update affect security in either a positive or possibly a negative way?
	Risks	What are the risks connected to the update?	What are the risks connected to the update?	What are the risks connected to the update?	What are the risks connected to the update?
	Customer acquisition		Does the update affect customer acquisition in any way?	Does the update affect customer acquisition in any way?	

*This column represents the categories on which decisions are made.

Figure 10 – Type of non-functional updates crossed against relevant factors of which the information can be found through asking the mentioned questions

5.1.1 Type of updates

This part provides a description of the four non-functional updates.

Language migration

This is, in most cases, the update that produces the most work out of the four updates. It can contain a conversion, re-architecting or rewriting of an application to a more modern programming language [35]. According to Gartner there are seven ways to carry out a language migration with different outcomes [36]. The drivers for initiating such updates are the potential benefits of creating less cost and business benefits i.e., increased customer satisfaction. Furthermore, it supports the goal of being on par with technological innovation. This can lead to another driver which is the employees.

The latter is what we have seen in practice. We have seen that the acquisition of a new development team was a driver (Interview G). Centric hired a new development team that consisted of young people. Because of the age the newly hired employees were more prone to modern languages than the ones that were being used.

However, the drivers are not always the triggers. We have seen an example of this within Centric, where a language migration update got triggered by the stop of support of a software vendor (Interview – G, H, I). This initiated the thought of change which led to the conclusion that the software was outdated, especially the front-end.

Upgrade

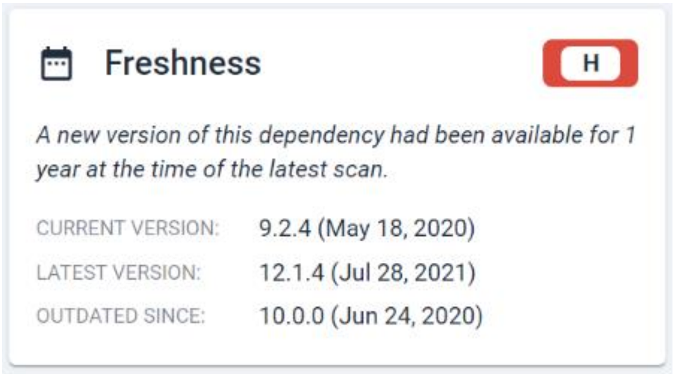
In this context the upgrade refers to a change to the major version when looking at semantic versioning. The two smaller ranks minor and patch version are not referred to [37].

One of the triggers of this update can be a dependency that requires it to be upgraded i.e., when it contains a breaking change. Another externally driven trigger could be that the older version will not receive support in the short future. A practical example seen during research was the internal development of common components to stimulate an upgrade elsewhere.

A driver to upgrade that got elicited from the data collection was to remove the 'technological barrier', i.e., always wanting to be up to date to reduce potential technical problems. The latest major version has the best processing power. Also, if the hosting of the concerning software is done elsewhere then being up to date ensures that there is support provided by the developing party. Other drivers of this specific update can be to improve the maintenance, security, and reduce the technical debt of the application in any way.

External library

Software engineers can choose to make use of external libraries when developing an application to reduce development-time and cost. This way the wheel does not have to be reinvented resulting in an increase of efficiency. However, the external party that produces and maintains the library is the party that updates the library [10]. This results in the using party to be dependent on the developing party. The software engineer that makes use of this library can retrieve the updates through the source they use e.g., GitHub. The mentioned software is publicly accessible as it is an open-source web-based interface. GitHub makes use of Git which is an open-source version control software [38].


Freshness H

A new version of this dependency had been available for 1 year at the time of the latest scan.

CURRENT VERSION:	9.2.4 (May 18, 2020)
LATEST VERSION:	12.1.4 (Jul 28, 2021)
OUTDATED SINCE:	10.0.0 (Jun 24, 2020)

Figure 11 - Example of an out-of-date external library from Sigrid

We have seen that, within Centric, a trigger for this type of update can be Sigrid. The system shows the freshness [39] of the external libraries. Freshness is a measurement of a single dependency showing the version difference between the latest version and currently used version. An example of a metric for measurement of freshness, that has been derived from the work of Cox et al., is libyear [11]. Not always does an outdated library form a trigger for an update. To stimulate this, policies on the maximum length of a library being outdated should be created.

Another trigger of updating library dependencies is that it is required when updating the related system. Most of the times when this is not done a breaking change will appear and a part of the system will not function. Through the interviews with the developers, we noticed that this is the most common trigger for updating external libraries.

Refactoring

Refactoring refers to processes that include changes done to a software system that do not affect the external behaviour, however it improves its internal structure [40].

This process has the goal to improve the maintainability, modularity, efficiency, reusability, reliability, extensibility, and complexity, i.e., the non-functional quality of the software. The goal of improving software quality can lead to a reduction of the amount of incoming technical debt created by the system [41].

5.1.2 Categories

This part covers the three categories that have been elicited through the data gathering.

Effort

This category will cover the criteria that are necessary to estimate the effort of the update.

Change size

How much of the code changes?

Estimating the size of the change is part of calculating the effort that is necessary to carry out the update. Getting the estimation can be done in different ways. It can be done by measuring the entire application and looking at the percentage of change that is necessary. Another option is just to check the amount that changes and create an estimation based upon past experiences and gut feeling.

The latter option can prove to be a time-efficient option whereas the first option is a more effective approach. Measuring the entire application has two main methods FPA (Function Point Analysis) [42] and counting the SLOC (Source Lines of Code) [39]. The FPA method has many different techniques developed by companies whereas SLOC is more straightforward.

However, the FPA focuses on user requirements as it calculates based on the functionality [43]. This perspective is less relevant in the context of non-functional updates. Next to that, according to Hira et al. [44] it is easy to calculate the SLOC after the deployment of software, which is almost always the case with non-functional updates.

Only looking at the SLOC can give an inaccurate view of the effort of the change. The reason for this is that counting the lines of code does not take the experience of a developer in the calculation. More experienced developers can create a 20% difference in code length [45]. Taking the approach of calculating the application first and looking at the percentage gives a more accurate result.

As stated by Fries the definition of a line of code is: "A line of code is any line of program text that is not a comment or blank line, regardless of the number of statements or fragments of statements on the line. This specifically includes all lines containing programming headers, declarations, and executable and non-executable statements. This is the predominant definition for line of code used today by researchers [46].

Breaking changes

Does the change introduce a break of some sort in a connected component?

Part of estimating the effort of the non-functional update is looking at the potential breaking changes. As the name gives away a breaking change is a change that results in some sort of break. The breaks that occur happen when a change to an API (Application Programming Interface) is made. The outcome of the break can vary. It can result in a minor inconvenience like a cosmetic effect, but it can also cause the application to fail [4]. A breaking change is in most cases not an additive change but relates to a modification or deletion.

Research done on the topic of the impact of breaking changes found that 6.8% of the major updates contain breaking changes. The definition of the major updates relates to semantic versioning [37]. This percentage translated to at least one breaking change each fifteen major updates. Although it does not seem to be an issue as major changes are not executed that frequently. However, in a company that has many software systems this is a significant number.

A system is either backwards compatible or it is not. When a system is backwards compatible it allows the use of data and interfaces from older versions of the same system [47]. This phenomenon enables to roll back the update when problems occur like breaking changes. When the system is not backwards compatible this needs to be taken into account with the effort of the update.

Dependencies

What are the dependent factors?

The third factor of getting an effort estimation are the dependencies of the software. This part relates to all the aspects the update is dependent on, including both technical and non-technical dependencies. The technical dependencies are the other software that rely on the concerning software. The non-technical dependencies are indirect technical dependencies. An example of this can be that the update requires another department to update their system first, resulting in being dependent on their roadmap.

The technical dependencies mostly relate to the external libraries that are connected through an API. This means that the update category ‘external library’ only relates to the non-technical dependencies. Recognizing the dependencies increases the awareness of the possible extra work like breaking changes.

Urgency

The following aspects will concern the urgency of executing a non-functional update.

Project support

Does the project for developing this product continue for a considerable time into the future?

This part only applies to the updates that are done for a software project or application that has an end date. Knowing the extent in which the project will continue provides a view on the end date of the project. When closing in on an end date it might not be beneficial to carry out the update.

This relates to the urgency since an update that would improve the product or project in the long run would be ineffective. The language update can use this factor the other way around. When a plan is made to have the product phased out in the next year it can function as a motivator to carry out the language migration update.

External support

Is there any indication that the supplier of the software has a motive of stopping support?

One of the aspects that can create urgency is the change in continuity of external software. This happens if a software vendor that creates and maintains software that the company uses decides to stop supporting the concerned software. The actions necessary to take depends on the part the software played. In some cases, the discontinuity of software support does not require any action, nevertheless it can cause urgency for a change.

Future innovations

Is the update still relevant in the future?

Rapid technological developments can impact the urgency of updates. This only plays a role for language migration updates due to the impact size. Trends like SaaS, when not implemented yet, can play a role in deciding the moment to carry out the update. A way to anticipate upcoming trends is to keep an eye out on the hype cycle created by Gartner. Also, as every part of technology evolves, the current perspective on what has a modern look and feel can change too. If one of the goals of the language migration is to create a more modern look and feel it must be taken into consideration how long this will stay modern. Reasoning behind this is that it might not be worth it if the look and feel must be changed after a short period again.

Market

Does the market in which the product resides impact the urgency in any way?

The market in which the application resides also plays a role in the urgency of the update. In less competitive markets the urgency of the update could have a lower urgency than in other markets with the same output.

Product Lifecycle

What phase of the product life cycle is the concerning product in?

The phase of the lifecycle where the product currently resides can play a role on the urgency too. A product that is in the maturity phase allows for an easier allocation of resources. One of the reasons for this can be that the application receives less customer requests, because it already is functionally rich enough. Looking at the earlier stages there are applications that find themselves to have a lower urgency, because the focus is on innovation. However, products that reside in the introduction or growth phase could also experience a high urgency if there is a chance that the update results in a competitive advantage like lower maintenance costs.

Current technical debt

How much TD has the product accrued over time?

As mentioned previously, shortcuts within software development can create technical debt. This debt increases overtime due to interest. Having knowledge of the current technical debt can play a role with prioritizing tasks like a refactoring update. When the software has racked up a significant amount of interest the urgency to carry out an update that reduced these increases.

Impact

The following five criteria give a view of the relevant aspects of non-functional updates when it comes to the impact. The first three have a business perspective and the last two have a customer perspective.

Future technical debt

What is the impact on the amount of accrued technical debt?

This part has a lot of similarities with the technical debt part in the urgency section. However, just as it can serve as an urgency it can also function as part of the impact of the update. Carrying out the update can impact the future level of technical debt. Having an estimate of this impact can provide an argument to prioritize the update higher.

Security

Does the update affect security in either a positive or possibly a negative way?

The impact the update has on the security is a crucial part to look at. The impact can be one of three options, negative, positive or neutral. When the update is expected to have a positive output, the decision could be made to carry out the update faster than expected before.

Customer acquisition

Does the update increase the possibility of selling the product?

The first aspect of the impact of the company side is the customer acquisition. This part brings up the question if the update helps in the procurement of new customers. Not only new customers can be affected but also the continuity of an old contract. An example of this aspect is the language migration of the frontend giving the product a better look and feel. This enhanced look and feel can help with customer acquisition.

5.2 Criteria collection

The criteria that have been elicited through the various data gathering methods are shown above. The place and manner in which they were found are shown in Table 4.

#	Criterion	Description
C1	Change size	This criterion got drafted after it became clear that effort was one of the aspects on which the decision is made. The code that needs to be changed is tackled with this criterion. We researched the way of measuring the change, the two methods that are regularly used are counting the SLOC and executing an FPA. When taking the context of non-functional updates being deprioritized into mind, makes the usage of SLOC in theory a better option. The advantage of using SLOC is that it takes up less time than using the FPA method. This criterion is not relevant for the external libraries as the developers do not change the code when carrying out the update.
C2	Breaking changes	Breaking changes is a criterion that got mentioned in the interviews a lot. The interviews in which it came forth were A, C, D and F.
C3	Dependencies	The dependency criterion got into the scope from the beginning of the research when there was talk about updates. We included a topic on dependencies into the interview protocol to confirm if this was an actual criterion that was involved in the decision-making process. The topic included a few questions to see in which manner the dependencies played a role.
C4	Project support	The project support criterion came forth out a mention made in interview E. The developer stated that he was working on the renewal of a product. This made us aware that there is also the possibility that software has an end date. Possible reasons for this end date could be that the output of the software is redundant, or it does not add value to the company.
C5	External support	The external support has been mentioned in the interviews C, D, F, G, H and I. The focus of the interviews G through I was mainly on the update type language migration. The interview questions were targeted at an ongoing example of the update type. In all three of the interviews, it was said that the external support was part of the trigger

		of the project. This makes it necessary information when evaluating the urgency of the updates.
C6	Future innovations	Another criterion that we retrieved from the second set of interviews was future innovations. One of the language migration updates encountered this criterion. If the update got carried out at a later moment, actions would be taken to also implement SaaS. However, this was not the case resulting in two separate projects which ultimately results in less efficiency. Also, interview H showed that the duration of how long it is perceived as modern should be taken into consideration. It was mentioned that a UI can be new for each four to five years. If this is the case and the project to realise the modernization of the product takes three years it might not be worth carrying out the update.
C7	Market	This criterion was obtained through interview G. The interviewee stated: “It could be that one of the domains/markets needs a decision or improvement less quickly than another”. This pointed at the fact that Centric delivers mostly in a public sector market and that the urgency of the update could differ depending on the market.
C8	Product lifecycle	In the second set of interviews we asked how the distribution of manhours is on maintaining the old product and working on the renewal of the new product. The response of interviewee H was: “25% of their time is reserved for maintaining the old product, because the product is 25 years old then so there was not much to add only to maintain”. Other interviewees responded similarly stating that the product receives less user requests as it is a mature product. This led us to adding the product lifecycle as a criterion in the decision process.
C9	Current Technical debt	From the beginning of the research the term technical debt was used. The same as the criterion dependencies this received a topic including questions. These questions functioned to see the level of involvement of technical debt within non-functional updates. The current amount of TD can provide an argument when prioritizing the update.
C10	Potential Technical debt	Knowing if the update has an impact on the amount of accrued technical debt can be a good argument when discussing its prioritization. Whether the update has a positive effect, no effect or maybe even a negative effect shows a part of the value of the update.
C11	Security	If the update has any impact on the IT-security, it should be known and taken into consideration when determining the impact.
C12	Risks	With every step taken it can prove useful to keep potential risks in mind. This is also the case with non-functional updates. This also got mentioned in interviews A, B and H.
C13	Customer acquisition	Knowing if the update influences the customer acquisition in any way can prove useful in the decision. Same as the potential TD criterion it provides a view of the value of the update. The criterion has been elicited from the interview H and I. They mentioned that the language migration helped provide a more modern look and feel to the product. The outdated look and feel had a negative impact on the customer acquisition.

Table 4 - Source of criteria

5.3 Requirements

Part of the methodology described by Wieringa [15] are the requirements of the design. This entails two tasks. The draft of these requirements, as shown in 2.2., and to see whether the model is able to meet the requirements. This part covers the latter task, the R# stands for the requirement number and D# for design.

R1: The artifact can be used by roles on multiple levels within the organization e.g., on operational level by developers or on tactical level by product owners.
D1: The model does not require information that only a certain role would know. This allows every role to make use of the model. Furthermore, the design does not make use of difficult language to ensure it is understandable for every user. Opening up the opportunity for discussion on the necessary information between roles. This stimulates collaboration between the various roles that are involved in the decision.

R2: The artifact should be able to be used for all the relevant non-functional updates
D2: The four non-functional updates that are identified are part of the same model. These updates are cross-referenced with the relevant criteria. This allows for every stakeholder to make use of the model. This possibly accelerates the improvements made to the model as it can be used for various scenarios. Next to that, it ensures that the model is used

R3: The artifact provides a way for the users to make data-driven decisions.
D3: The data gathering confirmed the assumption that currently the decisions are made on past experiences and gut feelings. The model provides information on the decision criteria that are partially already in use. Obtaining this information and acting on it should ensure more data-driven decision-making. It can also be the cause of discussion on which criteria are important in certain scenarios. This should next to the more data-driven part of it also support improved decision-making.

R4: The artifact provides an overview of decision criteria.
D4: The various criteria that have been gathered during the research are displayed in the left side of the model. After we discovered that the decisions are based upon three categories we decided to add these to the model. The criteria are subdivided under one of the three categories. This is to show the user on which part the criteria provides information. This allows the user to gather data for a specific category, providing a more efficient search.

Conclusion:
As described above, before any decisions could be made on the design of the model requirements had to be set. We came up with a set of four requirements, these were all met when looking at them in retrospect. This means that the model is fit for the job, if the right requirements were drafted. Arriving at the knowledge if this is the case can be seen from the results of it being used in practice. These results are leading and should indicate if the model actually is able to satisfy the needs.

5.4 Moment of updating

In order to convince other parties, substantiated arguments have to be made. This is especially the case when the party that needs convincing resides higher inside the organizational hierarchy. In project management this is usually done by making use of a business case. In case of carrying out an update it would be inefficient to create a business case. However, some sort of financial information on the topic can prove useful. This makes having knowledge on the moment of updating and the connected costs important.

The last knowledge question is about the most opportune moment to carry out non-functional updates. This question is a closed question causing it to contain a hypothesis. Our hypothesis is shown in figure 14. This shows a simplistic graph of the costs put against time. The graph shows that when immediately updating it resides on a high point of the costs. This is because keeping everything always up to date requires a lot of resources. The other highpoint of costs in the graph is on the maximum time passed. We rationalized that when waiting too long with non-functional updates the technical debt and its interest racks up the costs. Resulting in having an optimum T somewhere between the two extremes.

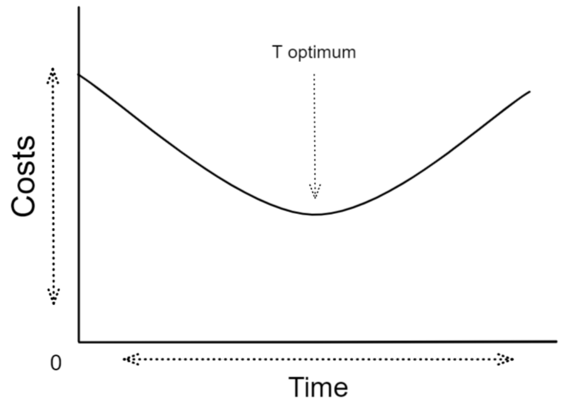


Figure 12 - Graph of costs versus time on moment of updating non-functional updates

Deciding when to allocate resources to carry out an update can be a difficult task. Part of the data collection was looking at the current thought process on this matter. As discussed in chapter 4.4, the approach taken for this was to provide the person responsible for choosing this scenario and ask for their thought process. The scenarios provided a way of asking product owners, amongst others, their process when choosing the moment of updating. This led to the finding that it contains uncertainty. The decision model provides an overview of the required categories and the underlying criteria.

A way to see, from a financial perspective, whether it is more profitable to carry out the update at a later moment is to calculate the costs and benefits of both timeframes. The main cost driver in both calculations will be the number of hours necessary. Within Centric the metric man-hour is used; this represents one hour of work of an employee. A metric that is widely used is full-time equivalent, it represents the time of a full workweek of an employee. We proceed to make use of the metric man-hours in this document. In the process of determining the effort of both timeframes the change of the code needs to be estimated. The number of man-hours for both timeframes need to be estimated based on the change size calculation. In the second set of interviews we added a question concerning the difference time makes on the number of man-hours necessary. The result of this was that they were under the impression that time would make no difference on the amount of man-hours. However, according to the second law of Lehman's laws of software evolution the complexity [25] of systems increases when a change is applied. This can cause an increase in labour that needs to be done for the update. Resulting in a difference of FTE when doing it now or in a later stadium.

Next to that, another aspect for the moment of updating that needs to be acknowledged is overspend due to technical debt. This acknowledgement of technical debt can be looked at from two different perspectives. First perspective is to look at the technical debt the update can resolve. Technical debt does not have to be a threat. However, not resolving the debt results in an increase of interest. The effect of the interest is shown in figure 7. Interest translates into additional money spent on maintenance because of technical quality issues [23]. This needs to be cut out in a timely manner as it allows the worth of the update to increase and consequently can be used as an argument to carry out the update sooner. Software that can calculate the technical debt of systems e.g., SonarQube and Coverity [48] can support estimating this part.

Furthermore, lost productivity is another aspect that is different between the time. The lost productivity relates to the work that could have been done by the resources that have been allocated to prepare the update. This attribute can be measured by dividing the output with the input and multiplying this by the number of manhours that could have been used for other projects. A different approach then reallocating resources is to hire additional personnel. However, this can greatly increase the cost.

Potential technical trends can also play a role in choosing the moment. This relates more to the larger updates such as language migrations. Through the interviews concerning the decisions we discovered that it would have been more efficient to wait with the update. After half a year of a multiple year project it became clear that SaaS was going to be used in the future. This caused for another project parallel to the already running project, in order to make the application SaaS ready. The hype cycle developed by Gartner could prove to be a fitting tool for this, trends that reside in the first phases need to be considered to have an impact.

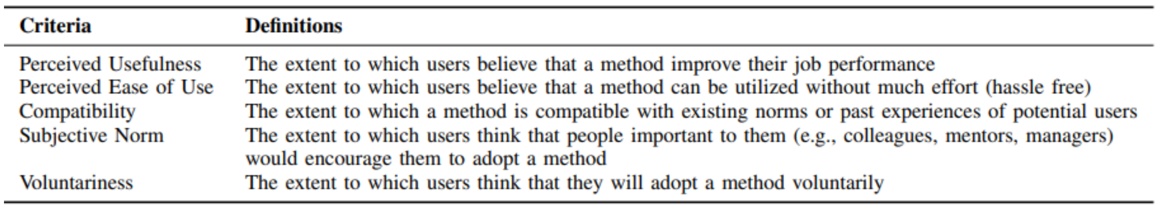
6. Acceptance of decision model within the case company

This part will discuss the method and results of the treatment validation.

6.1 Method of validation

The last step of the first design cycle is the treatment validation. The validation needs to be done before implementation; this is contradictory of the sequence in the engineering cycle. The treatment validation functions as an investigation to the effects of the artifact in a model of the problem context according to Wieringa [15]. Furthermore, these then get compared with the requirements and goals.

The methodology of design science states that the validation should consist of investigating interactions that occur between the prototype and the design in context. The investigation should be focused on the effects of this interaction. However, receiving the desired output for this with the current context would result in a time-consuming method. This could be done through monitoring the results over a period of time. The monitoring should provide information on the margin of error and extra time consumed on average by using the model. Because this method would take up a lot of time another method has been used. This method takes up less time and enables the potential users to provide their opinions on the matter. This type of user feedback can also stimulate the change process, if this is the case, opposed to the other method which would only supply raw data on the performance. Also, this route is more a form of evaluation than validation. The evaluation should happen after implementation as opposed to prior to it, taking on the sequence of design science.



Criteria	Definitions
Perceived Usefulness	The extent to which users believe that a method improve their job performance
Perceived Ease of Use	The extent to which users believe that a method can be utilized without much effort (hassle free)
Compatibility	The extent to which a method is compatible with existing norms or past experiences of potential users
Subjective Norm	The extent to which users think that people important to them (e.g., colleagues, mentors, managers) would encourage them to adopt a method
Voluntariness	The extent to which users think that they will adopt a method voluntarily

Figure 13 - Framework validation criteria [49]

The method we use to measure the level of acceptance derives from the method created by Riemenschneider et al. [50]. The methodology takes five criteria that are tested amongst the potential users of the artifact. Figure 13 shows the criteria and their definition, based on these criteria statements have been drafted. We have added one more criterion, correctness, to the already five existing ones. The correctness functions as validation on the elicited information.

The answers that can be given by the participants are partially pre-defined. The answers that can be provided are based on the six-point Likert scale [51]. The amount of answers dictates if the possibility for a neutral answer. The approach we use removes the neutral option as it provides six answers, this forces the participants to provide their opinion. To enable the participant to elaborate on their answer or provide additional remarks a section for comments is available below the statements. This can be seen in Appendix D.

Criteria	**Questions**
Usefulness	The model increases the performance of the method
Ease of use	The model is able to be used without using much effort
Compatibility	The model fits in with the current way of working
Subjective norm	The model is recommendable to colleagues
Voluntariness	I would use the model myself
Correctness	The model is without any faults and does not need additions

Table 5 - Statements concerning artifact validation

The artifact is an information providing support model. The model itself does not provide data, but it provides the concerning aspects. This way the user of the model must search the information on the specific update-related aspects. The success of the model goes hand in hand with the capabilities of the user. With this in mind, testing the acceptance of the model in the manner that has been described above would suffice to validate the model.

The selection of the stakeholders that is used to validate the acceptance is relatively effortless. This is due to the fact that the model can be used for all four of the non-functional updates. The people that were involved in prior inquiries are used for the validation. The questioned persons will have base information of the research and the artifact, making the information exchange more efficient.

Approaching and asking the concerning questions can be done in the context of the company's ways of working through two options. These two options were noticed during the ethnographic research that we carried out. The first option is providing the model combined with the questions through the chat function of Microsoft teams. The second option is inviting the stakeholders for a video conference in which the validation is done. The latter option provides a more effective approach since questions will be answered immediately. However, it is more time-consuming. This reduces the range of stakeholders reached, whereas the first option is less time-consuming resulting in a higher number of people that can be reached in the same timeframe. Also, all the answers are documented if answered through the chat function. With these arguments we choose to use the approach which makes use of the Microsoft teams chat function.

6.2 Validation results

The validations forms that have been distributed amongst the employees that were involved during the data gathering. The asked group consisted of three software developers, two management level employees and eight product owners. We received eight responses out of the thirteen total requests. The respondents were sent two different documents. The first document contained the statements on which the user could give their opinion, the second document provides the possibility for elaboration on the given answers.

The usage of the Likert scale for the statements allowed for a good presentation of the results, these results are presented in Figure 17. The results shown in the graph indicate that the respondents were mostly positive. However, the last statement concerning the correctness was perceived as less positive. The subjective norm that suggests that the model is recommendable to colleagues did not receive any sort of disagreement. Also, the statement that received the most strong agreements was the compatibility which relates to model fitting in the current way of working.

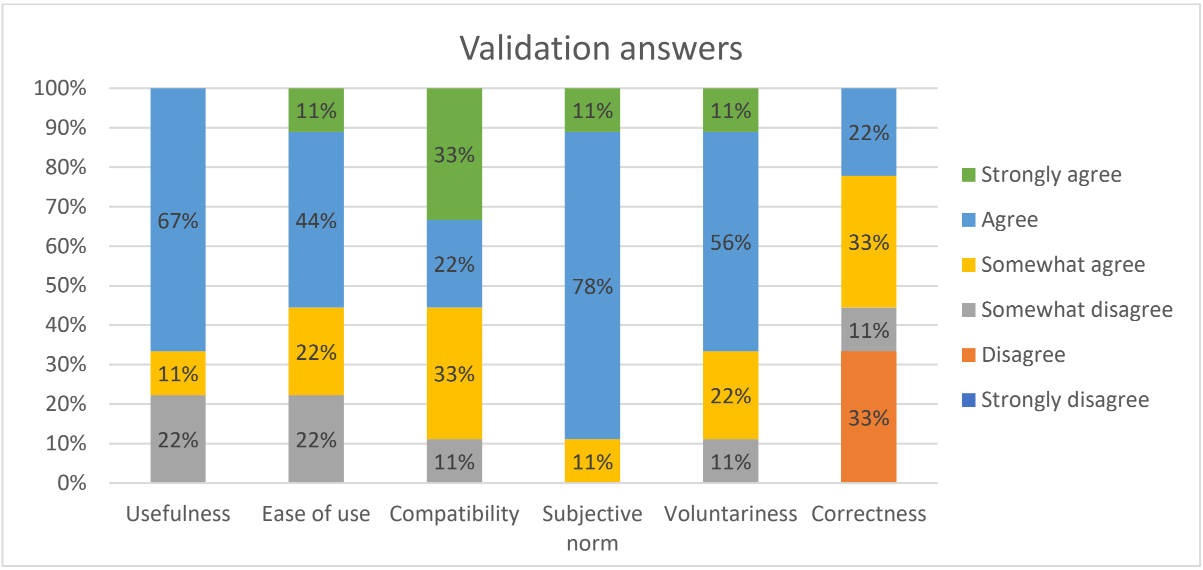


Figure 14 - Graph of the validation answers

The majority of the respondents provided comments alongside their opinion on the statements. Comments on the performance enhancing capabilities of the artifact swayed in a positive direction, this corresponds to the graph. But this was not the case with everyone, some stated that it was unclear if it would enhance performance. The graph of ease of use shows division in the answers. An opinion stated that the language used could be more effective when it consisted of less academic language and when adapted more to IT product domain language. Furthermore, an improvement would be giving an example of the type of updates. A respondent argued that the model reflects the current way of working resulting in a strong agreement on compatibility. Another argued that if it would fit perfectly into the way of working it would not deliver added value. The subjective norm received, as mentioned above, no disagreement and received no comments. The voluntariness of the respondent was dependent on which product is worked on. This is due to the fact that the supplier can be the party that hosts the technical side. The correctness received the least amount of agreement. Reasons for this given by the respondents was because it could be adapted into a more agile way, possibly through practice. Others stated that no model is without faults and that, as reality changes daily, it will likely be adapted to new developments and insights. The model having a supportive role could also be the cause of this as the outcome is dependent on the person using it. One respondent pointed out a fault of the model. This fault was a missing connection between breaking changes and refactoring.

In conclusion, the respondents were generally positive on the first five statements. The opinions diverged on the last statement, this statement received more disagreement than the others. This could be the cause of the respondents opinion being that the model is not without faults. It could also be the cause of a mismatch on what the respondents expect as outcome versus the desired outcome. This however can change once the model is in use and provides a different outcome than expected.

7. Discussion

This chapter will go over the earlier mentioned knowledge questions. It will also serve as a reflection on the results of the done research.

7.1 Knowledge questions results

Part of the methodology of design science that provided necessary direction were the knowledge questions. These allowed for a structured way of subdividing the necessary information that needed to be extracted. This section goes over the five different questions and their answers.

KQ1: Do all non-functional updates have the same decision process?

The goal of the first knowledge question was to examine if there was a difference in the decision process of the researched updates. If this was not the case, the design would not have to make a difference between the updates. The hypothesis for this knowledge question was that there would be a noticeable difference between the processes of the non-functional updates. We tested if this hypothesis was true through gathering data and creating an overview of the extracted data. We drafted a process model to gain this overview as shown in chapter 4.3. In the model it can be seen that the processes do differ on a few steps. However, the information retrieved from the scenarios combined with earlier obtained data presented us with the fact that the decision is dependent on three aspects. These three aspects are the categories that are used within the model: effort, urgency, and impact. The approach of acquiring the necessary data for these categories differs per update. The categories are split into criteria that have been obtained through the interviews. The criteria that are relevant for the concerning update can be elicited from the decision-model.

KQ2: What criteria are decisions of non-functional updates based upon?

To comprehend what information is necessary in order to make a data-driven decision we did research on the relevant criteria. The process of searching for these criteria continued throughout the whole research period. Every aspect of the data gathering added value to the criteria. The added value was not solely in the form of a new criteria, it could also be in the form of providing supporting arguments of the already identified criteria. The criteria that have been elicited throughout the research are shown below. Table 4 shows the criteria and the source.

KQ3: Who is responsible for choosing the moment of updating?

As Centric makes use of Scrum, an agile way of working, a possibility of deciding the moment to carry out updates can be done through prioritization. This does not provide a specific moment but does give a good indication. Once the update is prioritized in the upcoming sprint it provides the indication that the update will be carried out in one to four weeks. The mentioned range is the most common interval for sprints, how many weeks is dependent on the guidelines set by the company. As mentioned in chapter 4.2. the development team as a whole is responsible for prioritizing the updates. However, the role that is ultimately responsible for this decision is the product owner. The prioritization only applies as a form of deciding if the update can be covered by only one PBI. Once the update cannot be covered by a single PBI there are more roles responsible for the decision. This is the case with language migration updates. This type of update tends to be large, resulting the amount of work to be put into multiple PBI's. Due to it having multiple PBI's, a start date and a desired deadline needs to be assigned to ensure that the tasks do not get deprioritized. The decisions that we have seen within Centric were done by roles on strategic level. However, this conclusion is based on the type of updates done on large products. Small products or products early in the development require less effort which could alter the party that makes the decision.

KQ4: What are the triggers for non-functional updates?

During the search of the various triggers of non-functional updates we found that there is a distinction that can be made on the initiation of the trigger. The initiation of the trigger can either be actively searched for or it is a reaction. We used the terms proactive and reactive for this matter. The majority of the triggers that we elicited during the research are reactive, this is presented in Table 2. The lack of proactive triggers can be connected to the same reason that non-functional updates get deprioritized. This reason is that the output of the work does not provide the direct results that the alternative does. More information on the benefits of proactively searching for a reason to carry out non-functional updates can stimulate the search.

KQ5: Is there a most opportune moment to carry out non-functional updates?

Companies such as Centric could experience great value in knowing the most opportune moment of carrying out an update. This knowledge question was drafted to know if there was even such a moment. The hypothesis we drew on this question is that there is a moment which can be seen as most opportune, however this is not an exact moment as it relies on many aspects. The expectation of this is shown in Figure 12. In chapter 5.4 we discuss the various relevant factors when looking to update on different moments. The concerning factors are lost productivity, overspend, future innovations and complexity. The level of impact these factors have is unclear. These factors do not take into account the benefits or disadvantages of a timeframe. With the accumulation of the uncertainty of these factors we were not able to get a clear answer on this question. Even though we were not able to confirm or deny the hypothesis we are still behind the hypothesis.

7.2 Interpretation

The drafted knowledge questions made it clear which information is necessary in order to attain the desired design goal. Drafting these questions required to examine the problem and the context it resides in. The ethnographic research played part in discovering the knowledge that was necessary to create the design. The ethnographic research added little data to the study, however it provided insight on the problem context. This improved the data gathering, reducing off-topic questions about necessary background as this was already known. The first knowledge question shows that the four mentioned non-functional updates have different process steps. However, the difference in steps can be minor e.g. the language migrations adds a few more steps opposed to an upgrade. This insinuates that a model can support multiple type of updates even with the differences. Decisions on when to carry out the update mostly is done by product owners and his or her team. This is an exception with larger updates such as language migrations where roles on strategic level get involved. However, this does not mean that this is done the same in every company. We solely investigated Centric that makes use of the agile project management method scrum. The role that is responsible for this can vary if there is another project management method used. The one knowledge question without a solid answer is knowledge question five. A concrete answer to this would have provided great value to organisations that are uncertain about this topic. The most opportune moment depends too much on uncertainty of various aspects. This causes there to never be a clear answer on this question. However, when asking the question on a lower level an answer could be given. An example of this would be if the same question was concerning a certain type of update with similar situations. This could be tested by monitoring the outcome of using the model on a certain type of update and situation with at least two timeframes.

Reflecting on the study, we studied specific type of updates in the form of non-functional updates. These proved to be more difficult to describe to someone without knowledge on the study. However, describing the definition became clearer after attaching the two attributes. This can prove significant to the person that has to work with non-functional updates by being able to separate updates. Also, the study shows that the type of updates within the category non-functional updates can be combined into one model. This model shows, after the validation, that the design is not final. Future work can be done on the study but it does provide a basis of the subject.

8. Conclusion

The conclusion will include the limitation, possible generalizations and the answers on the knowledge questions combined with the contributions the artifact brings. Also, potential future work is mentioned in this chapter.

8.1 Research

The study started with getting a grip on the term non-functional updates. This did not seem like it was necessary or that it would be difficult in any way. After fiddling over many ways of explaining what was meant by the term and having discussions whether it was the right explanation we reached an understandable explanation. We separated the updates that effected the behaviour of software. Next to that the distinction based on urgency played a role in the definition. Updates that are mandatory should experience any deprioritization. The conducted interviews combined with the literature review acted as input for the design that improves the problem context. This improvement is done by providing a more data-driven decision making of carrying out non-functional updates. Substantiated arguments can be made to deprioritization any kind of updates due to this. The design we decided on was a model which supports decision making of non-functional updates. Initially the model had to provide a decision as output, however this turned out to be a difficult task. The model displays relevant decision criteria against the four elicited non-functional updates. The decision-making model helps, generally product owners, with prioritizing non-functional updates.

8.2 Limitations

The outbreak of the Covid-19 pandemic makes it more difficult reaching out to employees of Centric. Working from home removes the possibility of going to the office, resulting in communication and information transfer being solely online. Requests made face-to-face are way more likely to obtain a positive response [52]. This impacts the number of positive responses when requesting time of potential participants for either gathering data or validation.

Furthermore, a limitation of the research is the lack of previous studies on behaviour-preserving updates in general. The only type of update that gives a lot of results on Google Scholar is refactoring. This causes for a more difficult demarcation of the scope. This is due to the reason that the type of updates that fall within the category ‘behaviour-preserving’ are not stated in prior research.

Another limitation is the sample size for validating the decision-making model. The role, within Centric, for which the model provides the biggest value is product owner. Therefore, the opinion of the product owners on the matter is perceived the highest. However, only requesting product owners to participate in the validation results in a low response. This is due to the small number of product owners within the case company combined with possibility of a request being denied. The developers that were involved during the research process were also requested to participate. The research was already known to them, making it easier for them to understand the reason and goal. Having product owners as main target of the validation led to amass a few participants for the validation. However, this provided a diverse image, as the product owners all work in different teams on different products within the company. Uniform answer can lead to the conclusion that the model has the same output in all the different teams.

We decided to only use the population from within Centric. This could lead to a subjective view on the matter. Roles and the way of working can differ between companies, resulting in the possibility of a time-consuming process to find adequate participants elsewhere. Making it way more practical to look inside Centric for participants. Even though it is a limitation Centric is a large internationally active and well established company. This made us assume that the employees provide a sufficient reflection on the rest of the market. The company in which the data gathering found placemanages most of their libraries in-house. This does not mean that most of their libraries are entirely created internally. However, a small part of the library could be made in-house to accommodate the company's needs. This causes for a lack of use of libraries that are entirely made externally. This can lead to less findings and data gathering on what actions are taken when a vulnerability within the library is detected by maintenance software e.g. Sigrid. We noticed that few participants that were part of the data gathering were able to provide information on questions drafted about entirely externally created and hosted libraries.

As mentioned in the scope we cannot ensure that the type of updates labelled as non-functional updates do not contain any parts of behavioural change within them. In practice there is not actively worked on to separating the updates. This makes it difficult to ensure that the mentioned updates are without any behavioural changes.

The DS methodology of Hevner states [14]: "The results of the field testing will determine whether additional iterations of the relevance cycle are needed in this design science research project.". Pointing at the fact that once the design is finished and a form of field testing is executed, there is a possibility another walkthrough of the cycle is necessary to reach a desired output of the design. However, this research will only include one big cycle due to the time constraint.

In qualitative research, the perspective of the researcher has influence on how data is interpreted. Therefore, the possibility occurs that the thesis' findings can be biased.

8.3 Generalization

The conducted interviews with the developers indicated that not many libraries were entirely created externally. This became evident when questions like "How often are external libraries updated?" hardly received an answer. This shows that many libraries contain a part that is internally created within Centric. This could be the cause of two reasons. On one hand it can be done for security reasons. On the other hand, it could be done to accommodate specific needs. For the generalizability of the study, it is important that the software libraries do not deviate a lot from the software used in the same market. During the study we did not notice anything that would be different in similar companies. However, as mentioned own characteristics are provided to the libraries when internally created. We assume that those characteristics do not affect the gathered data on a large scale. With this we assume that the data and outcomes is generalizable..

8.4 The data gathering was done solely within Centric; a great part of their products focuses on the public sector market. We think that the data gathering provides us with a similar outcome as it would with data retrieved from companies focused on a different market. Resulting in added value for companies in any market. We make this assumption on the fact that Centric is a large company that is active internationally. This shows that the business operations are well round and have a high probability of having similarities with other companies.Contributions

The contribution of this thesis revolves around the non-functional updates. The search for the criteria that play a role in the decision when to carry out the updates led to an overview of criteria. These findings led to the shaping of a decision-model which is also a contribution of this research. It provides the users with acquiring potentially necessary data by searching answers to stated questions. The acquired data enables the users to make a more data-driven decision on when to carry out non-functional updates. Next to that, the criteria that are relevant when comparing different timeframes of carrying out the updates have been identified. This can function as preliminary research in future work on the topic of finding the most opportune moment to carry out the updates.

8.5 Future work

The research has a few points in which more research can be done. A few of the points mentioned in this chapter derive from the limitations mentioned in chapter 7.1. Others derive from points of interest which need more research.

The data used in the research came from within Centric, this company has a focus on the public sector market. The outcomes of the data gathered might be different when done in a different environment. An option to test this is by making use of the model in another environment, e.g.a highly competitive market when making a decision on a non-functional update. This can also help validate the part concerning the entirely external created libraries as the data gathered on this topic could prove to be too few. Information on this was, as mentioned in the limitation, a bit scarce resulting in necessary testing.

Additional data gathering can possibly be done on the decision-making on updates in general. In this research we were not able to ensure that the mentioned updates cannot be executed in combination with functional updates. In future work it might be beneficial looking at the decision making as a whole on updates. This way the benefits of carrying out non-functional updates could be displayed better when functional updates are also taken into consideration. Next to that, more data could be gathered on the benefits and ways of proactive triggering the process of non-functional updates. As described, bad code quality could cause a product to accrue a high amount of technical debt due to interest. Research on proactive triggers of non-functional updates could be a way of reducing the increase in technical debt and elevating the benefits of non-functional updates in the process. This can play a role in the prioritization of the updates. Furthermore, additional research on finding an answer to the knowledge question concerning the optimum moment of carrying out the updates. This could entail finding ways of calculating concerning aspects like overspend due to technical debt, increase in complexity and lost productivity. Moreover, it could entail finding a way to quantify the benefits of the timeframes.

Currently the model is validated by testing five acceptance criteria plus a criteria on the correctness. The acceptance criteria provide a good view on the acceptance. However, another method of validating the correctness could be applied. The form in which another method could be done is by using the model on past decisions of non-functional updates. The results when using the model could be set against the results of past decisions. This has not been done in this research due to time limitations. Executing this method of validating requires the search of historical data on past decisions. Also, it demands the search for a different employee with the same knowledge. On the topic of shortage of time another step that could be taken in future work is an additional walkthrough of the cycles. Hevner [14] states that the results of the field testing might indicate that there are additional iterations needed. The results of the validation done in this research indicate that not every participant agreed with the correctness of the model. With this in mind an additional walkthrough can improve the model.

Further research could also be done on providing the criteria with any sort of boundaries. This way the model can steer the decision in a certain path. An example of this could be that if the code change size exceeds 10% of the total code another role within the company has to oversee the decision as it is perceived substantial. There is a chance that the boundaries differ for each type of update which adds complexity to the task. Also, testcases could be drafted in which the model is used to see if there is a discrepancy in outcome of similar projects when making use of the model.

9. References

[1] J. Hayduk, “Internet and Human Capability: A Study In Parallel Evolution,” [Online]. Available: https://www.wired.com/insights/2014/11/internet-and-human-capability/.

[2] The Economic Times, “Definition of 'Software Maintenance',” [Online]. Available: https://economictimes.indiatimes.com/definition/software-maintenance#:~:text=The%20essential%20part%20of%20software,closer%20to%20the%20higher%20pole.. [Accessed 2021].

[3] A. Akbarzadeh and S. Katsikas, “Identifying and Analyzing Dependencies in and among Complex Cyber Physical Systems,” 1 March 2021. [Online]. Available: https://www.mdpi.com/1424-8220/21/5/1685. [Accessed 2021].

[4] T. Bush, “What Are Breaking Changes and How Do You Avoid Them?,” 19 November 2020. [Online]. Available: https://nordicapis.com/what-are-breaking-changes-and-how-do-you-avoid-them/.

[5] Reqtest, “Why is the difference between functional and Non-functional requirements important?,” 5 April 2012. [Online]. Available: https://reqtest.com/requirements-blog/functional-vs-non-functional-requirements/. [Accessed 2021].

[6] D. Radigan, “Story points and estimation,” [Online]. Available: https://www.atlassian.com/agile/project-management/estimation. [Accessed June 2021].

[7] H. Krasner, “The Cost of Poor quality software in the US: A 2018 report,” 26 September 2018. [Online]. Available: https://www.it-cisq.org/the-cost-of-poor-quality-software-in-the-us-a-2018-report/The-Cost-of-Poor-Quality-Software-in-the-US-2018-Report.pdf. [Accessed 2021].

[8] O. B. Capers Jones, in *The Economics of Software Quality*, 2011, pp. 360-390.

[9] E. E. Ogheneovo, “On the Relationship between Software Complexity and Maintenance Costs,” 20 October 2014. [Online]. Available: https://www.scirp.org/html/1-1730107_51631.htm?pagespeed=noscript#ref14. [Accessed 2021].

[10] S. Raemaekers, A. van Deursen and J. Visser, “An Analysis of Dependence on Third-party libraries in open source and proprietary systems,” 11 February 2015. [Online]. Available: https://www.researchgate.net/profile/Joost-Visser/publication/265673176_An_Analysis_of_Dependence_on_Third-party_Libraries_in_Open_Source_and_Proprietary_Systems/links/54dbc3530cf2a7769d92fd6d/An-Analysis-of-Dependence-on-Third-party-Libraries-in-Open-Sou. [Accessed 2021].

[11] J. Cox, E. Brouwers, M. v. Eekelen and J. Visser, “Measuring Dependency Freshness in software systems,” 17 August 2015. [Online]. Available: https://repository.ubn.ru.nl/bitstream/handle/2066/143749/143749.pdf?sequence=1&isAllowed=y. [Accessed 2021].

[12] “Semantic Versioning 2.0.0,” [Online]. Available: https://semver.org/.

[13] A. Bhattacherjee, Social Science Research: Principles, Methods, and Practices, 2012.

[14] A. R. Hevner, “A Three Cycle View of Design Science Research,” in *Information Systems and Decision Sciences*, Scandinavian Journal of Information Systems, 2007, pp. 87-92.

[15] R. J. Wieringa, Design science methodology, New York, Dordrecht, Lodon: Springer Heidelberg, 2014.

[16] M. jackson, “Software Requirements and Specifications: A Lexicon of Practice, Principles and Prejudices,” 1995. [Online]. Available: https://www.amazon.com/Software-Requirements-Specifications-Principles-Prejudices/dp/0201877120. [Accessed 2021].

[17] S. Gregor and A. Hevner, “Positioning and Presenting Design Science Research for Maximum Impact,” June 2013. [Online]. Available: https://www.researchgate.net/profile/Alan-Hevner/publication/262350911_Positioning_and_Presenting_Design_Science_Research_for_Maximum_Impact/links/55802a8b08ae87edac4c8f60/Positioning-and-Presenting-Design-Science-Research-for-Maximum-Impact.pdf. [Accessed 2021].

[18] [Online]. Available: https://www.atlassian.com/software/jira.

[19] J.-L. Letouzey and D. Whelan, “Introduction to the Technical Debt Concept,” May 2016. [Online]. Available: https://www.agilealliance.org/wp-content/uploads/2016/05/IntroductiontotheTechnicalDebtConcept-V-02.pdf. [Accessed 2021].

[20] S. McConnell, “Managing Technical Debt,” 1 June 2008. [Online]. Available: http://www.construx.com/uploadedfiles/resources/whitepapers/Managing%20Technical%20Debt.pdf. [Accessed 2021].

[21] E. Tom, A. Aurum and R. Vidgen, “An exploration of technical debt,” 21 January 2013. [Online]. Available: https://www.sciencedirect.com/science/article/abs/pii/S0164121213000022. [Accessed 2021].

[22] B. Curtis, “Estimating the size, cost, and types of Technical debt,” June 2012. [Online]. Available: https://www.researchgate.net/profile/Bill-Curtis-2/publication/261464425_Estimating_the_size_cost_and_types_of_Technical_Debt/links/5707c65508aea6608133191f/Estimating-the-size-cost-and-types-of-Technical-Debt.pdf. [Accessed 2021].

[23] A. Nugroho, J. Visser and T. Kuipers, “An empirical model of technical debt and interest,” 23 May 2011. [Online]. Available: https://dl.acm.org/doi/10.1145/1985362.1985364.

[24] E. d. S. Maldonado, R. Abdalkareem, E. Shihab and A. Serebrenik, “An Empirical Study On the Removal of Self-Admitted Technical Debt,” 22 September 2017. [Online]. Available: http://das.encs.concordia.ca/uploads/2017/07/maldonado_icsme2017.pdf. [Accessed 2021].

[25] J. Visser, “The World Is Eating Your Software,” 24 September 2019. [Online]. Available: https://www.cutter.com/article/world-eating-your%C2%A0software-504691. [Accessed 2021].

[26] M. M. Lehman, “Laws of Software Evolution Revisited,” 10 June 2005. [Online]. Available: https://link.springer.com/chapter/10.1007%2FBFb0017737. [Accessed 2021].

[27] R. G. Kula, D. M. German, A. Ouni, T. Ishio and K. Inoue, “Do Developers Update Their Library Dependencies?,” February 2018. [Online]. Available: https://www.researchgate.net/publication/316920992_Do_developers_update_their_library_dependencies. [Accessed 2021].

[28] A. Zerouali, E. Constantinou, T. Mens, G. Robles and J. Gonzales-Barahona, “An Empirical Analysis of Technical Lag in npm Package Dependencies,” April 2018. [Online]. Available: https://www.researchgate.net/profile/Ahmed-Zerouali-2/publication/324149427_An_Empirical_Analysis_of_Technical_Lag_in_npm_Package_Dependencies/links/5c1b7dbca6fdccfc705aeb29/An-Empirical-Analysis-of-Technical-Lag-in-npm-Package-Dependencies.pdf. [Accessed 2021].

[29] [Online]. Available: https://libyear.com/.

[30] D. Parmar, “Speed of Technology: The Business Cost of Keeping Your Software Updated,” 6 July 2016. [Online]. Available: https://www.business.com/articles/speed-of-technology-the-business-cost-of-keeping-your-software-updated/. [Accessed 2021].

[31] M. Weber and T. Langer, Rational decision making, 2010.

[32] F. C. Lunenburg, “The decision making process,” 2010. [Online]. Available: http://nationalforum.com/Electronic%20Journal%20Volumes/Lunenburg,%20Fred%20C.%20The%20Decision%20Making%20Process%20NFEASJ%20V27%20N4%202010.pdf. [Accessed 2021].

[33] E. Blair, “A reflexive exploration of two qualitative data coding techniques,” 19 January 2015. [Online]. Available: https://journals.uair.arizona.edu/index.php/jmmss/article/viewFile/18772/18421. [Accessed 2021].

[34] [Online]. Available: https://www.maxqda.com/.

[35] Castsoftware, “Software Modernization,” [Online]. Available: https://www.castsoftware.com/glossary/software-modernization.

[36] S. Moore, “7 Options to Modernize Legacy Systems,” 14 December 2020. [Online]. Available: https://www.gartner.com/smarterwithgartner/7-options-to-modernize-legacy-systems/.

[37] S. Raemaekers, A. van Deursen and J. Visser, “Semantic Versioning and Impact of Breaking Changes in the Maven Repository,” 20 February 2016. [Online]. Available: http://pure.tudelft.nl/ws/files/19482719/TUD_SERG_2016_011_cc.pdf.

[38] Digital.gov, “An Introduction to GitHub,” 14 July 2020. [Online]. Available: https://digital.gov/resources/an-introduction-github/. [Accessed 2021].

[39] J. Cox, E. Bouwers, M. van Eekelen and J. Visser, “Measuring Dependency Freshness in Software Systems,” [Online]. Available:

https://repository.ubn.ru.nl/bitstream/handle/2066/143749/143749.pdf?sequence=1&isAllowed=y.

[40] W. F. Opdyke and R. Johnson, “Refactoring Object-Oriented Frameworks,” July 1992. [Online]. Available: https://www.researchgate.net/publication/2498589_Refactoring_Object-Oriented_Frameworks.

[41] Z. Codabux, “A Quality Assurance Approach to Technical Debt,” July 2014. [Online]. Available: https://www.researchgate.net/publication/269337986_A_Quality_Assurance_Approach_to_Technical_Debt.

[42] Ifpug, “ABOUT FUNCTION POINT ANALYSIS,” [Online]. Available: https://www.ifpug.org/about-function-point-analysis/.

[43] H. v. Geerubgeb, “Software size measures and their use in software project cost estimation,” 18 May 2015. [Online]. Available: https://nesma.org/2015/05/software-size-measures-and-their-use-in-software-project-cost-estimation/.

[44] B. B. Anandi Hira, “Function Point Analysis for Software Maintenance,” 8 September 2016. [Online]. Available: https://dl.acm.org/doi/abs/10.1145/2961111.2962613.

[45] E. Weiss, “Measuring LOC and other,” March 2002. [Online]. Available: https://files.ifi.uzh.ch/rerg/amadeus/teaching/seminars/seminar_ws0203/Seminar_3.pdf.

[46] R. C. Fries, “Establishin and using metrics,” in *Reliable Design of Medical Devices*, London, New York, Taylor & Francis, 2005, p. 290.

[47] TechTarget Contributer, “backward compatible (backward compatibility),” August 2005. [Online]. Available: https://whatis.techtarget.com/definition/backward-compatible-backward-compatibility.

[48] N. Choksi and C. DeArdo, “How to measure – and manage – technical debt,” 6 July 2020. [Online]. Available: https://enterprisersproject.com/article/2020/7/technical-debt-how-measure#:~:text=Tools%20such%20as%20SonarQube%20and,problematic%20code%20into%20quality%20code..

[49] A. Aldris, A. Nugroho, P. Lago and J. Visser, “Measuring the Degree of Service Orientation in Proprietary SOA Systems,” March 2013. [Online]. Available: https://www.researchgate.net/publication/234167870_Measuring_the_Degree_of_Service_Orientation_in_Proprietary_SOA_Systems. [Accessed 2021].

[50] C. K. Riemenschneider, B. C. Hardgrave and F. D. Davis, “Explaining software developer acceptance of methodologies: a comparison of five theoretical models,” December 2002. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/1158287/authors#authors. [Accessed 2021].

[51] G. Albaum, “The Likert scale revisited: an alternate version,” April 1997. [Online]. Available: https://fac.ksu.edu.sa/sites/default/files/likert20.pdf. [Accessed 2021].

[52] V. Bohns, “A Face-to-Face Request Is 34 Times More Successful Than an Email,” 11 April 2017. [Online]. Available: https://hbr.org/2017/04/a-face-to-face-request-is-34-times-more-successful-than-an-email. [Accessed 2021].

[53] A. R. Hevner, “A Three Cycle View of Design,” 2007. [Online]. Available: https://www.researchgate.net/profile/Alan-Hevner/publication/254804390_A_Three_Cycle_View_of_Design_Science_Research/links/0c96053b4a9f2d7adc000000/A-Three-Cycle-View-of-Design-Science-Research.pdf.

[54] “Source Lines of Code,” 15 July 2013. [Online]. Available: https://pvs-studio.com/en/blog/terms/0086/.

10. Appendix

10.1 A: Interview protocol

Interview protocol

Introduction

<I am currently working on my thesis within Centric. The thesis is focused on investigating the possibilities around making the moment of updating more fact-based. A part of the gathering the data is carrying out interviews. That is also the reason for sending this message. I would like to ask if there is a person within the Tiber team which I can conduct an interview with?>

Topic	**Question**	**&***
Introduction	Can you tell me a bit about yourself?	
Follow-up questions	Can you tell me about the product you work on?	
	What is your function within Centric?	
General	Who is the end responsible for updating the product?	
	What is the team composition?	
	Which products/team are you a part of?	
	What is your role within these teams/products?	&y
Updating	What are non-functional updates within the products you work on?	
	Could you give me examples of triggers of non-functional updates?	
	Do you do regression testing?	
	Did the regression ever point out a problem of the update within the system?	&y
	If PO: How do you do the prioritization? Is this done on gut feeling or data?	
Technical debt	Are you familiar with the metaphor technical debt (TD)?	
	Are there updates which get pushed back the prioritization even though they are important?	&n
	Do you have insight on the TD off the product(s)?	&&y
	Do you measure it yourself too?	&y
	Based on what criteria do you measure it?	&
Person specific questions	<The amount of TD on the software they work on>	
	<Their plan on reducing it>	
	<Questions, if present, about their vulnerable dependencies>	
	<Project specific question>	
Problem statement	Has there been a time where complications started after the update was completed?	
	Has there ever been a moment where an update was forced by anything?	
	How many hours a week/month do you use for updating non-functional related components? (costs)	
	Could you take me through the process of a non-functional update?	
	Has it occurred that unsolved bugs create more problems? (Impact of unsolved bugs)	&
	Do you make use of automated pull request?	&
	How are pull requests currently done? Are you familiar with Dependabot?	
	What happens when you wait with bug fixes and focus on other things?	
Dependencies	Does the product make use of external libraries?	
	How often are these external libraries updated?	&y
	Who makes the decision for this?	&
	What are the triggers for updating the external libraries?	
	Have there been problems when updating external libraries?	
	What happen when an external library contains a breaking change?	

Table 6 - Interview protocol

***& =** Dependency of the question above. The output of the question above determines the flow. y stands for a question answered with yes and n with no. Every extra '&' implies the follow up answer is related to the question another row up.

10.2 B: Coded segments of first set of interviews

Code	***Coded segments***
Technical Debt\Time / problem	The only question is when there will be time free to do these things, this depends on the flow of the projects etc. Transcription #2 - Key2Begraven: 21 - 21 (0) We want to increase our SIG score, to keep up. But sometimes we do not have time for it. Transcription #2 - Key2Begraven: 53 - 53 (0)
	We have asked if we were able to make a distinction but this was a limitation to the system or it costed too much to invest into Transcription #5 - Key2Financien: 58 - 58 (0) We are also under time pressure Transcription #5 - Key2Financien: 70 - 70 (0)
Technical Debt\Reason	My experience is that it has to do with agreements with the customer or when at the start of the project or when something is bought. Some of the functionalities have to be delivered on a certain moment. This set moment can be the cause of the shortcuts which create TD. Transcription #2 - Key2Begraven: 13 - 13 (0)
	But this created some technical debt because we hurried some functionality, made it inefficient or simply made it not in the acceptance criteria that we worked on before Transcription #3 - Key2Belastingen: 43 - 43 (0) We wanted to do it sooner not in the next month, but we understood that the client now has some breaking changes that need to be done Transcription #3 - Key2Belastingen: 119 - 119 (0)
	It is a bit difficult with SIG because PL/SQL is not their strong suite Transcription #5 - Key2Financien: 52 - 52 (0) This can be because the investment towards fixing technical debt is not really profitable because the project stops in three to four years Transcription #5 - Key2Financien: 62 - 62 (0)
Technical Debt\Approach	try to be more proactive than reactive Transcription #1 - Privacy workspace: 17 - 17 (0) discuss it within the team, to see what we can improve, and do we improve and how much time it will take because its relevant. Transcription #1 - Privacy workspace: 30 - 30 (0)
	Well, every idea that I have and that other have will be put on the backlog. That is basically where I would put the output of a sprint. Next to that I keep track of a prioritization of the things on the backlog. You can also see it as following, take for example that I see a piece of technical debt or something that is useful for us to handle. That is also on the list and I prioritize that. Transcription #2 - Key2Begraven: 21 - 21 (0)
	First we looked at the code self to see the entire development part and looked at the components. And so, what kind of changes we would expect to make and how to make those. So we had a plan in mind what to use how to use and when to use, then we already had the experience of testing the components of the previous sprints Transcription #3 - Key2Belastingen: 47 - 47 (0) I would honestly start with the one of our most inefficient pages

	Transcription #3 - Key2Belastingen: 53 - 53 (0)
	besides the regular PBI's we try to create PBI's for us where we try to refactor our components, remove duplications, refactor because we need to fix the code quality Transcription #4 - Styleguide: 38 - 38 (0)
	So, when we have a planning. We create those PBI's because we push for those and we talk to the PO saying that we kind of need it. Do you think we have time for this we ask, and he says well let's see. Then we create the PBI's we estimate them and when we plan a new sprint we take them into account, we check the effort so there is time specially reserved for this kinds of tasks. But if there are smaller things that we can do to increase quality then we can take these as we go without planning to much. Only the big components need to be discussed and planned properly. Transcription #4 - Styleguide: 42 - 42 (0)
	Take for example the example I had given with the big procedure. If I check it and I see that there are good sub-procedures then I say that it is not necessary. Transcription #5 - Key2Financien: 56 - 56 (0)
	We have nowadays within a sprint a PBI with the task to check and if possible to improve the SIG score of the touched code Transcription #5 - Key2Financien: 64 - 64 (0)
	The PO trusts our ability to estimate that Transcription #5 - Key2Financien: 76 - 76 (0)
Technical Debt\Amount of TD	six days of technical debt Transcription #1 - Privacy workspace: 14 - 14 (0)
	three days of technical debt Transcription #4 - Styleguide: 40 - 40 (0)
	on SonarQube I have seen that the backend has 23 days of technical debt Transcription #5 - Key2Financien: 53 - 53 (0)
Technical Debt\Measuring	No, not really. Not on the way that it should be kept. I do have an overview, but it is an overview for myself. It is not recorded somewhere. Transcription #2 - Key2Begraven: 9 - 9 (0)
	Own interpretation Transcription #2 - Key2Begraven: 11 - 11 (0)
Technical Debt\Measuring\Tool	we use sonarqube Transcription #1 - Privacy workspace: 12 - 12 (0)
	we can apply the SIG rating to grow our score Transcription #3 - Key2Belastingen: 51 - 51 (0)
	So we do monitor SonarQube and SIG statistics Transcription #4 - Styleguide: 38 - 38 (0)
	We get rapports from SIG Transcription #5 - Key2Financien: 52 - 52 (0)
	SIG

	Transcription #5 - Key2Financien: 54 - 54 (0)
	This has to do with the fact that SonarQube can read Angular code Transcription #5 - Key2Financien: 70 - 70 (0)
Technical Debt\Are you familiar with technical debt	Yes Transcription #1 - Privacy workspace: 8 - 8 (0)
	Yes Transcription #2 - Key2Begraven: 6 - 6 (0)
	You mean SiG Transcription #2 - Key2Begraven: 51 - 51 (0)
	Yes Transcription #3 - Key2Belastingen: 41 - 41 (0)
	Yes Transcription #4 - Styleguide: 36 - 36 (0)
	Yes Transcription #5 - Key2Financien: 50 - 50 (0)
Why did you get room to lower technical debt	I can not say for sure if yes, they allowed it for their goodness of their heart or they did it because they didn't have anything more urgent that would have pushed it away Transcription #3 - Key2Belastingen: 117 - 117 (0)
Dependencies	It actually has only one dependency Transcription #1 - Privacy workspace: 48 - 48 (0) we hope that they never change and when they change practically it should be a reactive approach Transcription #1 - Privacy workspace: 58 - 58 (0) four vulnerability dependencies Transcription #1 - Privacy workspace: 67 - 67 (0) On the 245 there are three with a high risk and one with a medium risk Vulnerabilities shown on SIG Transcription #1 - Privacy workspace: 69 - 69 (0) Sigrid accepts it as a vulnerability but in reality, it is not a vulnerability Transcription #1 - Privacy workspace: 71 - 71 (0) so this tool is practically scanning everything in the library, but you might not use everything from that library Transcription #1 - Privacy workspace: 76 - 76 (0)
	needed to be changed twice or thrice because the requirements of the customer changed as well, and the current package did not respect those requirements Transcription #3 - Key2Belastingen: 105 - 105 (0)
Dependencies\Planning to reduce vulnerable dependencies	We read the release note, see what the change is see what the advantage is of the newer version is of the library and maybe estimate how much time it will take and discuss with the product owner. If it is

	really easy to implement a breaking change, we do not have to discuss with the product owner. I mean if it is 30 minutes, we do it Transcription #1 - Privacy workspace: 60 - 60 (0)
	So, looking at the stars from GitHub for example is very important. You do not take something which has 10 stars on GitHub. It means that it has been liked by 10 people. Transcription #1 - Privacy workspace: 62 - 62 (0)
	At the moment, no. I expect that when the time will come, and we will have more budget allocated on privacy workspace we are going to start working on it. That is also the time when we are going to slowly update the packages again. Transcription #1 - Privacy workspace: 78 - 78 (0)
	Its more possibly up to gut feeling or that we know that we will update one, then we will update them all Transcription #3 - Key2Belastingen: 97 - 97 (0)
	So, those are kind of the triggers, maybe a related package has been updated or when you have time for it Transcription #3 - Key2Belastingen: 102 - 102 (0)
Dependencies\Update frequency	We update that as often as possible if it doesn't have breaking changes Transcription #1 - Privacy workspace: 58 - 58 (0)
	I think the last time that we updated them was six or seven months ago Transcription #3 - Key2Belastingen: 97 - 97 (0)
	We update them together with the angular version Transcription #4 - Styleguide: 82 - 82 (0)
Updating\Breaking changes	When one of us does the update locally before implementing it further, we usually see in the changelog of that given package how many breaking changes there are and what the changing components require to change are. Transcription #3 - Key2Belastingen: 70 - 70 (0)
	They say that the component that we previously used is going to use a different set of data, if you update it this is going to break. For the external packages we have to locally test the entire package. Transcription #3 - Key2Belastingen: 74 - 74 (0)
	right now we cannot update the latest version of typescript. Even typescript themselves tell us to not update typescript Transcription #3 - Key2Belastingen: 107 - 107 (0)
	we try to be as careful as possible because we known major version from angular contain breaking changes Transcription #4 - Styleguide: 58 - 58 (0)
Updating\Prioritization	Yes so I basically prioritize them based on what the customer reports Transcription #2 - Key2Begraven: 17 - 17 (0)
	its mostly gut feeling Transcription #3 - Key2Belastingen: 39 - 39 (0)
	we as a team then discuss them prioritize them and get to fixing them

	Transcription #4 - Styleguide: 19 - 19 (0)
	we have a rating from one to four Transcription #4 - Styleguide: 29 - 29 (0)
	we get together as a team and discuss the request and the PO takes in our opinions into account and takes a final decision Transcription #4 - Styleguide: 34 - 34 (0)
	So the one to four is also based upon if the software that is using it is critical. Transcription #4 - Styleguide: 73 - 73 (0)
	This is basically decided by the PO in conversation with the architects Transcription #5 - Key2Financien: 34 - 34 (0)
	So it is based on a market research and what the consultants know about the needs of the customers. Transcription #5 - Key2Financien: 40 - 40 (0)
Updating\Trigger	Most probably a problem Transcription #1 - Privacy workspace: 54 - 54 (0)
	to many bugs in a component Transcription #1 - Privacy workspace: 54 - 54 (0)
	from my point of view, is that we are going to receive a ticket from one of our customers that it is not working in a certain browser or something. Transcription #1 - Privacy workspace: 82 - 82 (0)
	We have arranged it in a way so that the customer sends an incident through the customer portal, which is picked up by us. We judge if it really is an incident and if it is what kind of incident it is Transcription #2 - Key2Begraven: 15 - 15 (0)
	the ones that the client wanted in the project Transcription #3 - Key2Belastingen: 29 - 29 (0)
	Mostly the triggers are from the testers who test the software and see if there is a bug or something that needs to be fixed and, in the end, there are the consultants that check if everything is good. Transcription #3 - Key2Belastingen: 34 - 34 (0)
	When the Styleguide has support for angular 11, then we transition to angular 11 as well Transcription #3 - Key2Belastingen: 125 - 125 (0)
	It’s a combination between, we know the version is out so there is a bit of pressure to be up to date and there is also pressure from the clients that they want to update Transcription #4 - Styleguide: 50 - 50 (0)
Updating\Non-functional updates	That are technical updates. Code standards, best practices those are the most important Transcription #5 - Key2Financien: 22 - 22 (0)
Updating\Non-functional updates\Time consumed	zero Transcription #1 - Privacy workspace: 50 - 50 (0)
	But it doesn’t take that much time now that I think of it

Transcription #3 - Key2Belastingen: 76 - 76 (0)

The update didn't take that long, it took a week

Transcription #4 - Styleguide: 56 - 56 (0)

I think that that we reserve 8 hours a week for this.

Transcription #5 - Key2Financien: 24 - 24 (0)

We have a meeting with the three of us which takes an hour once in the three weeks.

Transcription #5 - Key2Financien: 26 - 26 (0)

Updating\Problems

the time is very limited and mostly we do not do anything

Transcription #1 - Privacy workspace: 22 - 22 (0)

so the way that that we are separated, backend basically updates itself and we do not have to make any changes. So it takes us by surprise sometimes, because there are sometimes unexpected breaking changes. That sometimes stops us from working on a given module

Transcription #3 - Key2Belastingen: 59 - 59 (0)

If the next big angular update we are going to do will acquire a new, newer version of typescript that we know causes breaking changes

Transcription #3 - Key2Belastingen: 113 - 113 (0)

we don't know the end users opinion at all, at the moment

Transcription #4 - Styleguide: 23 - 23 (0)

We actually wanted to update to 11 a lot sooner, but we also have to upgrade typescript to typescript 4

Transcription #4 - Styleguide: 58 - 58 (0)

We did have trouble with the order of the PBI's. Because the backend should be ahead of the frontend development, because otherwise they have to do redundant work since they will need to get mock data and after that they have to use the real endpoints

Transcription #5 - Key2Financien: 42 - 42 (0)

Updating\Problems\Fix approach

If we cannot fix the problem one option is to roll back the changes

Transcription #1 - Privacy workspace: 44 - 44 (0)

We solved it and we learnt from it. This led to our approach being more phased and we rollback faster if there are uncertainties.

Transcription #2 - Key2Begraven: 31 - 31 (0)

we will sound the alarm and say that we need time to fix this

Transcription #3 - Key2Belastingen: 113 - 113 (0)

Updating\Problems\Problems after update

Yes certainly

Transcription #2 - Key2Begraven: 37 - 37 (0)

Updating\Problems\Problems during update

yes

Transcription #1 - Privacy workspace: 44 - 44 (0)

We walked through the steps we usually take, but apparently there was a step that we missed

	Transcription #2 - Key2Begraven: 31 - 31 (0)
End responsible	But there is the project manager which will make the biggest decisions, blame or compliments of that project Transcription #3 - Key2Belastingen: 25 - 25 (0)
	the PO has the final word Transcription #4 - Styleguide: 13 - 13 (0)
	The PO together with the development team Transcription #5 - Key2Financien 25 - 25 (0)
Team composition\Other projects	we work on six different products Transcription #1 - Privacy workspace: 28 - 28 (0)
	I have around four hours of time a week that I reserve for the old application Key2Finance Transcription #5 - Key2Financien: 20 - 20 (0)

Table 7 - Coded segments retrieved from interviews

10.3 C: Second set of interviews example

Interview: Project #1
Date: 17-06-2021
Interviewer: Jonathan Cammeraat
Interviewee: Product manager #1

Topic	Question	E or C*
Keuze	- Wat is de trigger geweest om over de transitie na te denken in eerste instantie? o Waarom de keuze om op dat moment het Yellowstone project te starten? o Wat zijn de voordelen van ORDS tegenover Oracle forms naast modernere look?	E E E
	- Wat was de doorslaggevende factor om voor Angular te kiezen boven ASP.net? o Waren de extra aantal uren de doorslaggevende factor?	E C
	- Was er geen nieuwe/andere versie van Oracle forms dat ook voor een modernere view zorgde?	E
Time	- Wat is de deadline/ eindplan? o Zou dit het benodigde tijdsbestek anders zijn als er op een eerder of later moment het project gestart was?	E E
	- Zijn mogelijke complicaties ook meegenomen in de berekening van het aantal uur? Of wordt hiernaar gekeken tijdens het project?	E
	- De kosten bestaat volledig uit de geschatte benodigde uren. Zijn de gemaakt uren de enige kostenpost bij dit project? - Wordt bijvoorbeeld de verloren productiviteit ook meegenomen of de extra benodigde trainingen?	C E
Extra	- Aangegeven in de PAR staat dat jullie verplicht waren om eerst een on premises oplossing te realiseren, zou het efficiënter zijn geweest als het in een keer naar SaaS kon en niet eerst via on-premise? o Hoe lang zou er nog gewacht moeten worden voordat er in een keer naar SaaS gegaan kon worden?	C E
	- Wordt het oude product nog onderhouden? o Door hoeveel mensen wordt dit ondehouden? (Dat nog gebruikt wordt neem ik aan? Of is het al gereleased?), want iedereen is nu aan t ombouwen?	E E
	- Wanneer was er gekozen om in de toekomst gebruik te maken van het nebula platform als een SaaS-oplossing? - Toen was het al duidelijk dat Key2Burgerzaken over moest, waarom is toen de transitie niet gedaan? - Wat is nog benodigd om over te gaan naar SaaS als het yellowstone project klaar is?	E C E
	- Zijn er negatieve effecten van de verandering?	E

Table 8 - Interview protocol second set

* E = Exploratory question
C = Confirmatory question

10.3.1 English example

Interview: Project #1
Date: 17-06-2021
Interviewer: Jonathan Cammeraat
Interviewee: Product manager #1

Topic	Question	E or C*
Choice	- What has been the trigger to think about the transition in the first place? o What was the reason to start the project {project} on that moment? o What are the other benefits, next to a more modern look, of ORDS opposed to Oracle forms	E E E
	- What was the deciding factor to choose Angular above ASP.net? o Were the extra hours detrimental in the decision?	E C
	- Was there not a newer or different version of Oracle forms that created a more modern look and feel?	E
Time	- What is the deadline of the project? o Would the necessary time to complete the project be different when the project started earlier or later?	E E
	- Are there any complications connected to the calculation of the number of necessary hours? o Is this something that is looked at during the project?	E C
	- The costs that are made are solely based upon hours made. Are these the only type of costs for this project?	C
	- Is the lost productivity something that is taken into account?	E
Extra	- In the PAR it states that you were forced to realize the solution on premises first, would it have been more efficient if there was a possibility of going SaaS immediately? o How long would have been necessary to wait in order to go SaaS in the first go?	C E
	- Is the old product still getting maintained? o If yes, how many people maintain it? o If no, is everyone who used to maintain it being used in completing the project?	E C C
	- When was chosen to make use of the SaaS platforms? o Was it not clear at that point that {product} had to do the transition? o What is still necessary to be done in order for {project} to go over to a Saas-solution?	E C E
	- Are there any negative effects connected to {project}?	E

Table 9 - English version of second set interview protocol

* E = Exploratory question
C = Confirmatory question

10.4 D: Validation form

Validation form

Below are six statements shown that can be answered with six different answers. The answers for each statement should be chosen from table 2. Below the tables is a section provided to add extra comments if necessary. The first table is equipped with a reference row, this can be used to refer to in the comments.

Reference	Statements	Answer
A	The model increases the performance of the method	
B	The model is able to be used without using much effort	
C	The model fits in with the current way of working	
D	The model is recommendable to colleagues	
E	I would use the model myself	
F	The model is without any faults and does not need additions	

Table 10 – Validation statements

Strongly disagree	Disagree	Slightly disagree	Slightly agree	Agree	Strongly agree
1	2	3	4	5	6

Table 11 - Likert scale answers

Comments:

10.5 E: Validation response

#	Role	Comments
1	Business development manager	-
2	Product owner	"The model more or less reflects what we are doing already (so 6), albeit more or less unconsciously. therefore, I would certainly give it a try some time, but am not sure if I would continue (so a 4)."
3	Developer	-
4	Tech lead	-
5	Developer	"**A**: I assume this means that decision making and planning process around non-functional updates, will go better/easier/faster if we use the model." "**C**: If the model would completely fit in with the current way of working, one could argue that it has no added value to the current way of working" "**F**: Haha. Nice try.. No model is without faults, and apart from that, more than likely it will be adapted to new developments and insights, as our reality changes daily."
6	Developer	"This table can be helpful in evaluating a change" "Once I will have more decision power in a project, I might use this or something similar to assist me."
7	Product owner	"Having some criteria in place would be very useful when deciding priorities based on the type of change involved, especially a table with things to check, so you will not miss to take into consideration a category or an important criterion."
8	Product owner	"Having less academic language and adapted more to the language used in the IT product domain and company (e.g. Language migration can be easily confused with Upgrade – it's hard to draw the line between them without going through the explanations)"
9	Product owner	"Comment E: I don't think our department would have much added value because our products are built on SAP's technology services. For us, it's more about functional content. Technical topics are taken over by SAP." "Comment F: It's a good start - but I think every model will have to prove itself in practice and should be adapted in an agile way."

Table 12 - Comments of validation forms