MASTER'S THESIS

Improving information architecture by documenting domain knowledge in conceptual models

How software creation can benefit from leveraging conceptual models

Huiting, J. (Jos)

Award date:
2020

Improving information architecture by documenting domain knowledge in conceptual models

How software creation can benefit from leveraging conceptual models

Student:	Jos Huiting
Identiteitsnummer:	851105875
Datum rapport:	18-02-2020
Datum presentatie:	03-03-2020
Datum einde inschrijving:	18-09-2020

Improving information architecture by documenting domain knowledge in conceptual models

How software creation can benefit from leveraging conceptual models

Opleiding:	Open Universiteit, faculteit Management, Science & Technology Masteropleiding Business Process Management & IT
Programma:	Open University of the Netherlands, faculty of Management, Science & Technology Master Business Process Management & IT
Cursus:	IM9806 Afstudeertraject Business Process Management and IT
Student:	Jos Huiting
Identiteitsnummer:	851105875
Datum:	18-02-2020
Afstudeerbegeleider	Stef Joosten
Meelezer:	Rogier van de Wetering
Versie nummer:	1.0
Status:	definitief

Abstract

In the last decade, Agile has become the default software development process. This paper questions if parts of the traditional system analysis process could still be valuable in the context of an Agile process. This paper is focused on conceptual models which can share domain knowledge, by documenting in text or a visual representation. The aim of this thesis is to find out if a conceptual model brings value in the software development process, or that it is an avoidable effort?

The research question that was formulated: *Does the usage of conceptual models with domain knowledge lead to a better understanding of the subject domain?*

An experiment has been conducted which included two groups that had a limited amount of time to study a significant amount of domain knowledge. One of the groups received a conceptual model and a text to explain the domain, while the other groups just received the text.

The groups of participants that were included in the experiment did not have significant differences in properties amongst them. The result of the experiment was that there was no significant difference for either of the categories of questions, as well in the overall results.

Keywords

Software creation, conceptual models, domain knowledge, information architecture, shared understanding, knowledge gap

Summary

Introduction
In the last decade, the process of software engineering changed dramatically. Agile took over the software development world by storm, and most companies in practice use it as their default methodology these days. For both business and technical teams, the rapid iterations with a lot of communication have advantages compared to the lengthy, traditional processes. This method ensures that small increments are validated and quickly delivered to the business, which replaces the waterfall of processes that were part of the technical and functional design phases in traditional approaches. A great example is the system analysis phase, which is defined as a transfer and transformation process in which domain specialists need to transfer their knowledge to the requirement engineers (Olmos & Rodas, 2014). This paper questions whether parts of the system analysis process could still be valuable in the context of an Agile development process. This paper focusses on conceptual models in particular, which can be leveraged to establish a shared domain understanding between the business and software teams. Conceptual models can help to share knowledge, by documenting in text and sharing a visual representation. Maximizing the amount of domain knowledge, that is shared within a team, should ultimately result in a better information architecture of the project. The gap in domain knowledge between the business and the software engineering team is a key factor in project failures. Some researchers even think that fixing this gap is crucial for the survival of organizations. The aim of this thesis is to investigate whether a conceptual model brings value in the software development process.

Therefore, the following research question was formulated: *Does the usage of conceptual models, visualizing domain knowledge, lead to a better understanding of the subject domain?*

Methods
In this thesis, the additional value of the usage of a conceptual model has been investigated within an experiment, that tested the understanding of a certain subject domain in two groups of software engineers; one group with a conceptual model, and one group without a conceptual model. Both groups had a limited amount of time to study a significant amount of domain knowledge. One of the groups received a conceptual model and a text to explain the domain, while the other groups only received the text. After internalizing the knowledge, the participants finished a questionnaire that consists of multiple-choice, problem-solving questions and a Cloze test. The total score was calculated as an average of these three tests. Both the scores for parts as well as the total score for the tests are recorded as results.

Results
The groups of participants *(N=23)* that were included in the experiment did not have significant differences in properties amongst them. The result of the experiment was that there was no significant difference for either of the categories of questions, as well in the overall results.

Discussion
Within this experiment, no additional value of conceptual models was found. One possible explanation is that most of the value of a conceptual model is in the process of creating it, as it might be the case that most of the domain knowledge transfer happens in that stage. One limitation of the current setup of the experiment with UML models was the limited capability of the models to express knowledge. The knowledge models that are used contained basic capability to express knowledge as entities and relationships. However, compared to more formal approaches, that e.g. use relational algebra, the certainty that knowledge is expressed correctly in the model, is limited. We recommend conducting a next experiment based on such a model, to further validate the usage of conceptual models.

Given the importance of the shared understanding of the domain, we recommend for teams in practice to experiment with various models to transfer domain knowledge and increase conceptual knowledge in a team. This could take the shape of the conceptual model but could also be documentation or regular domain-related discussions with stakeholders. We advise to find a way that helps to tighten this knowledge gap, on both the conceptual level as well as the technical level.

Conclusion

Within the experiment of this thesis, the additional value of a conceptual model for improving the understanding of the subject domain could not be shown. However, based on the importance of shared understanding of the domain, we recommend for teams in practice to experiment with various models to transfer domain knowledge and increase conceptual knowledge in a team.

TABLE OF CONTENTS

1. INTRODUCTION 7

1.1 BACKGROUND 7
1.2 RELEVANCE 7
1.3 RESEARCH OBJECTIVE AND QUESTIONS 9

2 THEORETICAL FRAMEWORK 9

2.1 LITERATURE RESEARCH 9
2.2 IMPLEMENTATION 10

3 METHODS 11

3.1 CONCEPTUAL DESIGN: SELECT THE RESEARCH METHOD(S) 11
3.2 TECHNICAL DESIGN: ELABORATION OF THE METHOD 12
3.2.1 EXPERIMENT 13
3.2.1 SCORING SYSTEM 15
3.2.2 PRIMARY AND SECONDARY ENDPOINTS 15
3.3 STATISTICS 15

4. RESULTS 16

4.1 DEVIATIONS FROM THE PLAN 16
4.2 EXPERIMENT - PARTICIPANTS 16
4.3 EXPERIMENT - RESULTS 17
4.3.1 MULTIPLE CHOICE QUESTIONS 17
4.3.2 PROBLEM SOLVING QUESTIONS 18
4.3.3 CLOZE TEST 19
4.3.4 SCORE MODEL 20
4.3.5 OVERALL RESULTS 20
4.3.6 EXPLORITORY RESULTS - SYSTEM MODELING EXPERIENCE 20

5 DISCUSSION 21

5.1 DISCUSSION - PARTICIPANTS 21
5.2 DISCUSSION - RESULTS 22
5.3 DISCUSSION – LIMITATIONS & REFLECTION 23
5.4 CONCLUSIONS 24
5.5 RECOMMENDATIONS FOR PRACTICE 25
5.6 RECOMMENDATIONS FOR FURTHER RESEARCH 25

REFERENCES 26

ATTACHMENT 1 – EXPERIMENT TEXT 28
ATTACHMENT 2 - EXPERIMENT MODEL 29
ATTACHMENT 3 - EXPERIMENT QUESTIONS AND ANSWERS 30

1. Introduction

1.1 Background

This thesis finalizes my Master of Science degree in Business Process Management & IT at Open Universiteit. The main purpose of this thesis is to investigate if we can improve the process of building software by incorporating more domain knowledge. In particular the question whether domain knowledge improves the design of systems that solve complex business problems in close collaboration with the business. In the last decade a lot of new methods have emerged in the software development space, like Domain Driven Design (Evans, 2004) and Event Sourcing (Fowler, 2011). Such methods are indented to bring software closer to the business, and model it more similar to the processes in the domain compared to traditional software development. This allows domain experts to contribute in an early stage of the process and should therefore affect the outcome positively. This raises the question how this approach can be incorporated in the widely used agile software development (ASD) methodology (Cockburn & Highsmith, 2001). Compared to more classic, plan-based approaches, Agile has fewer formal requirements engineering phases and replaces this with ad-hoc conversations. This results in lot of small feedback loops instead of lengthy processes of upfront design. The question remains whether these conversations are an effective way to transfer domain knowledge between domain experts and software engineers, and if this can be improved.

These questions also relate to the problems we are solving with my employer, Mendix. Mendix has built a low-code platform that greatly helps business and IT to work closely together and quickly deliver applications that fulfill the needs of the business. I am interested in how to improve this collaboration by adding the possibility to share domain knowledge more explicitly. My goal for this research is to gain more insight into how business and IT can better collaborate on the knowledge required to build software. This should contribute to the existing body of knowledge on this subject.

1.2 Relevance

One of the important aspects of a software project is to understand the problem space of the information system that needs to be built. A significant amount of time of the project is spent in understanding the domain of the project. In traditional software methodologies, this phase is known as the system analysis phase (Davis, 1988). In recent days, this phase is commonly referred to as Requirements Engineering (RE). RE is concerned with interpreting and understanding stakeholders terminology, viewpoints, and goals (Nuseibeh & Easterbrook, 2000). It is a transfer and transformation process of knowledge in which domain specialists need to transfer their knowledge to the requirement engineers so that they can design a set of requirements (Olmos & Rodas, 2014). This set of requirements specifies the demands the software has to meet, which subsequently determines what knowledge is needed to complete the project successfully. This knowledge can also be used retrospectively to verify whether the software adheres to the specification (Olmos & Rodas, 2014). To document the domain knowledge of the stakeholders, a model can be used that supports RE-activities (van Lamsweerde, 2002).

Rubin identified two types of models; *system* and *conceptual* models (Rubin & Rubin, 2011). A *system model* can document various views on a technical system and is most naturally represented by the source code of an application. Therefore, most projects do not contain additional documentation for the system model. *Conceptual modeling* is described by Mylopoulos as “activity of formally describing some aspects of the physical and social world around us” (Mylopoulos, 1992). The conceptual model can be used for “purposes of understanding and communication”, which is confirmed by Rubin & Rubin as they see constant communication as the core purpose of conceptual models (Rubin & Rubin, 2011). To more clearly scope the definition of a conceptual model, we use the definition by Gemino (Gemino & Wand, 2005): Every model which is indented to define and share domain knowledge with shareholders we see as a conceptual model. This enables shared

understanding and agreement on the domain. A conceptual model is a communication tool used in the early phase of design to let stakeholder's communication about requirements and systems design. Therefore they need to be understandable for stakeholders, analysts and designers (Gemino & Wand, 2005).

Shared conceptualization is a deeper understanding compared to the shared understanding as described by Extreme Programming (Abdullah, Honiden, Sharp, Nuseibeh, & Notkin, 2011). All individuals within the team share a common understanding of the product requirements, which supports the requirement activities (Abdullah et al., 2011). This information is not stored in story cards but in a shared mental model by the actors in the team. When we consider that teams constantly collaborate to update their shared mental model, the question rises what exactly is stored in such a model. A significant part of this shared model is domain knowledge. This perspective on the requirements engineering process is clearly defined by Olmos & Rodas, who describe a requirements engineering as "a process that transfers domain knowledge into specifications" (Olmos & Rodas, 2014). Figure 1 shows the various actors that are involved, and their contribution to the specifications that are the result of this process. Specifications are properties and behavior a program has, based on the demands by the domain. These specifications are used to build software. The required domain knowledge is split into *tacit* and *explicit* domain knowledge (Olmos & Rodas, 2014). Tacit knowledge is "knowing more than we can tell" (Polanyi, 1967). This is the kind of knowledge which domain specialists have but which is not captured in any form of documentation. This is a hidden side of individual knowledge stored in the mind, beliefs and thoughts which is reflected in various kind of actions and commitment (Nonaka & Takeuchi, 1995). Hackney describes tacit knowledge as knowledge that exists in the minds of people but which is hard to communicate about, both verbally as well as written (Al-Salti & Hackney, 2011). An urgent need to Sharing this knowledge in agile organisations and teams crucial for survival of organisations (Ichijo & Kohlbacher, 2008). Wang identifies a gap in knowledge in knowledge in projects, which significantly contributes to project failure (Wang, Chia-Lin Lin, Jiang, & Klein, 2007). Amos see issues on team performance, innovation and efficiency as well which ultimately can lead to failure of a project (Ambos, Ambos, Eich, & Puck, 2016; Haldin-Herrgard, 2000). Glinz categorizes shared understanding a bit differently, in explicit shared understanding (ESU) and implicit shared understanding (ISU) (Glinz & Fricker, 2015). ESU is about explicit specifications such as requirements, documents and manuals. ISU is the common understanding of non-specified knowledge, assumptions, opinions and values. Glinz confirms that conceptual models can be useful in transferring knowledge, as it helps building an explicit shared understanding of the domain (Glinz & Fricker, 2015).

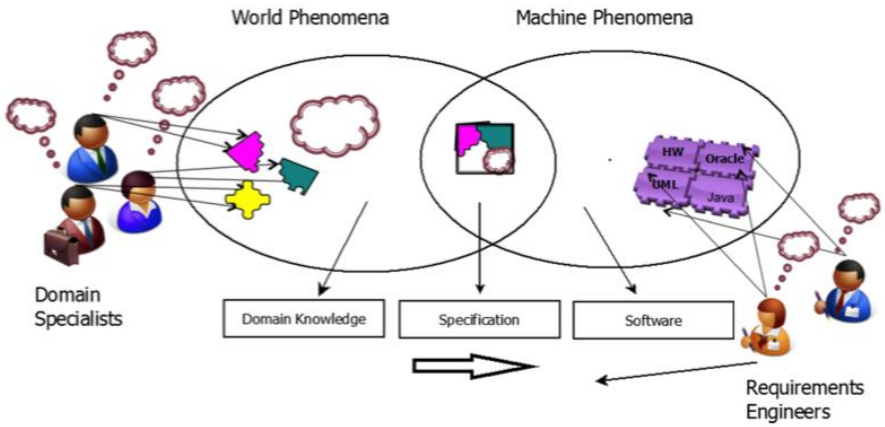


Figure 1: Requirements engineering perspective based on knowledge. The puzzle pieces represent explicit knowledge and the clouds tacit knowledge. It describes transforming domain knowledge to a software design (source: *Olmos & Rodas, 2014).*

According to the literature the knowledge transfer can be improved by introducing conceptual models to facilitate the conversations. In practice most Agile projects do not use a conceptual model,

but some still deliver successfully. This raises the question whether conceptual modeling is still necessary. Or is it an avoidable effort? This research attempts to clarify this matter by assessing the effect of conceptual modeling in agile teams. Even when we second the agile credo "*The most efficient and effective method of conveying information to and within a development team is face-to-face conversation*" (Beck et al., 2001), we are still curious to see whether conceptual modeling still has a place in the ASD-world.

1.3 Research objective and questions

The objective of this research is to investigate if there a conceptual model is an effective way for teams to transfer knowledge. To be more precise, the transfer and conversion of tacit and explicit knowledge. This is really important in the context of agile development because it replaces a lot of the up-front design from more classic development approaches. Conceptual models can help to share knowledge, by documenting them in text and sharing a visual representation. So, a conceptual model can help to maximize the amount of knowledge that is shared explicitly amongst a team. The outcome of the experiment should gather data about the value of conceptual models, to use it for agile development teams. Aside from improving specifications when building software these models can also prove useful for onboarding team members and handovers between teams.

Problem statement: Team performance in agile software projects is negatively affected by the tacit knowledge gap. Does documenting domain knowledge in conceptual models contribute to closing this gap?

For practical reasons, we want to minimize the result and avoid an endless transfer of tacit knowledge. Therefore, we limit the conceptual model to things that have consequences in the system to be built.

Research question: *Does the usage of conceptual models with domain knowledge lead to a better understanding of the subject domain?*

Hypothesis: The usage of a conceptual model leads to a better understanding of the subject domain.

2 Theoretical framework

2.1 Literature Research

In practice, we observed that it is challenging to make information tangible enough to enable teams to build a correct information system. The information architecture is an important step to transfer the required knowledge and enables engineers to the systems that adhere to the requirements by the business. As agile engineering practices are becoming more commonplace, information modeling seems to become more and more obsolete. This raises the question if capturing knowledge in a model can help to make the information architecture more concrete. In this section, we explore some of the questions that we faced while researching this question.

Iterative and incremental software development methodologies are commonly used in the industry these days. Most popular is the agile software development (ASD) methodology, which name originates from the publication of the agile manifesto, which contains the core values of agile development (Beck et al., 2001). Agile methods include feature-driven development, Extreme Programming and Scrum (Dybå & Dingsøyr, 2008). ASD-based methods rely on constant collaboration between developers and users, small increments, and frequent deployments by a single small team with end-to-end responsibility (Abrahamsson, Warsta, Siponen, & Ronkainen, 2003). This collaboration is also noticed in RE-activities of ASD-based projects. Requirements are gathered during the whole project, instead of one structured phase at the beginning of the project

(Schön, Thomaschewski, & Escalona, 2017). Stakeholders, software developers and end-users work closely together on an ad-hoc basis to refine the requirements to a level that they can be used for development tasks (Schön et al., 2017). These requirements are validated using stories, up-front test cases and early and continuous validation (Glinz & Fricker, 2015). It seems that conceptual models models are not accurately represented anywhere, and when the domain evolves these models might get lost in ASD-based projects (Rubin & Rubin, 2011).

Therefore, the question rises which knowledge we do need to share in the context of an agile project. The notice of a shared mental model is also mentioned in Agile methodologies, e.g. in Extreme Programming. Shared understanding is a crucial prerequisite for the successful development and deployment of any software system (Glinz & Fricker, 2015; Hoffmann, Bittner, & Leimeister, 2013; Tan, 1993). A team that has a shared understanding of the requirements has an understanding which is shared as a mental model (Buchan, 2014). The team should execute activities that identify gaps in understanding of the requirements and keep fixing them by agreeing on a new version of this understanding. The aim is to have a consistent understanding of requirements across the team, which is documented in a conceptual model.

What is a conceptual model? In paragraph 1.3 the conceptual model was introduced and and narrowed it to the definition by Gemino (Gemino & Wand, 2005). A conceptual model should enable shared understanding and agreement on the domain. This domain is something that matters in the real word, e.g. the domain of a certain business. Any knowledge model can be used as a conceptual model for the purpose of this research, provided that it has the possibility to express rules. We restrict this to have the ability to use a conceptual model as the source of truth and document then definition of the knowledge of a business model. There are various ways to document conceptual models. To document a model in an explicit and unambiguous way it can be formalized in business rules, e.g. in Ampersand. Ampersand defines a conceptual model as a collection of concepts, relations, and rules that define the domain language (business language) of the problem at hand (Michels, Joosten, Van Der Woude, & Joosten, 2011). Another way to express conceptual models is by leveraging the Unified Modeling Language (UML), which in my observation is well-known amongst practitioners. However, there is some discussion around which conceptual models semantics can be expressed using UML. Partridge recognizes two categories, concepts tied to a mind and concepts that are mind-independent (Partridge, Gonzalez-Perez, & Henderson-Sellers, 2013).

2.2 Implementation

Due to the broad scope of the literature search I had problems at first to clearly set a scope for my research. I reviewed about 70-80 papers to determine relevancy and whether or not they are worth reading. In the beginning I read in a broader scope to gain knowledge on the subject matter, in later phases of the research I zoomed in on the subjects that were more relevant of the research. While reading I determined usefulness based on the abstract, and sometimes a scan of the literature section of the paper.

The first iterations to answer the question involved exploratory research to discover what kind of terms are related to the topic of knowledge models. Most of this research involved terminology like “knowledge model”, “information architecture,” “agile” and “modeling.” The outcome of this initial search was that the subject was quite broad, and research was conducted from fields like computer science, requirements engineering, business informatics, and psychology.

The main question that arose after these searches were the role that knowledge plays, and how that can be improved in agile projects. This question narrowed down to the knowledge, which is required to design software systems, and in particular what kind of knowledge was required from the business. When the research shaped towards the current subject the search revolved around terms

like “tacit knowledge”, “shared understanding”, “mental model team”, “knowledge management” and “agile”. To get more specific results most of the combinations of these terms where combined. If too many results showed up the results were limited by providing the correct field, where “computer science” was mostly used. Quite some of the research around knowledge transfer and understanding was done from the perspective of psychology, so that was added as a filter as well.

In the final stretch of the research it narrowed down to the transfer of knowledge in the contact of Agile teams, and what kind of knowledge should be transferred. This led to questions like what “implicit shared understanding” is, and how it influences the way software is built. During the setup of the experiment there has been some research as well, which helps to understand how to measure “conceptual models understandability” as a concept.

Selection criteria have been applied to ensure that the selected literature was of the right quality. The criteria used included the fact that it was it involved peer reviews, impact factors and the usage of just published papers. Right now, around 45 different sources of literature are used in this paper. All of these pieces of literature are either relevant for the introduction of the subject matter, as well as the method of research. The literature search website of the OU, *bibliotheek.ou.nl* was mostly used for literature search. To widen the range of results *Google Scholar* was checked as well.

3 Methods

3.1 Conceptual design: select the research method(s)

The objective of this research is to determine whether conceptual models can contribute to improving the information architecture to enhance the shared understanding of domain knowledge in agile teams to improve the information architecture of the project. In particular the part where these models help to make implicit domain knowledge explicit. Glinz also states that domain knowledge reduces the probability that software engineers misinterpret specifications or fill gaps in the specification in an unintended way (Glinz & Fricker, 2015). If shared knowledge can be consolidated in a conceptual model, such a model might be a valuable asset in an agile team for preventing misunderstandings and enabling precisely formulated agreements that ease domain knowledge transfer.
In our research we make the assumption that an increased understanding of the domain by an agile team will improve the information architecture. Based on this we want to investigate if a conceptual model can be leveraged by practitioners to achieve a better shared understanding of the domain. This improved shared understanding should be perceived by practitioners. To measure the understanding by team members, we need to measure the quality of the mental representation of the domain.

In general, there is a wide array of methods in research that can be used, both qualitative and quantitative methods. Saunders mentions that this separation between those methods is not that strict in practice, as research design is likely to combine quantitative and qualitative elements (Saunders, Lewis, & Thornhill, 2008). This research purposes to combine these, to increase validity and more accurately measure the understanding of the participants. In agile teams a lot of the knowledge transfers happen as part of conversations. So, we could leverage the conversations themselves to research this topic. However, just observing the conversation would suffer from some variables that are difficult to control. This would yield mostly qualitative data by the observer, instead of measuring the actual understanding by the participants. Another problem with this approach is reproducibility, as it does not yield a result that is easily repeatable and verifiable. It really depends on the people that are in the research cluster, their willingness to have a discussion and quality of the input. It also would be a challenge that the outcome will be affected by the

observer effect, which would make it less reliable (Spano, 2005). To counter this, we choose to focus on the outcome of the communication in the design process, which is documented in a conceptual model. Key in this is measuring understanding, which is required to validate if a participant has gained certain knowledge around a topic. The conceptual model that is used needs to be translated into a mental model, which is used to understand certain information. We can illustrate that by an example in the mathematics field. The sentence "A linear regression is a model that established the relationship between dependent variable y and independent variable *x"* is difficult to understand if you don't have a previous knowledge, hence no mental model around this subject. You will need to have an understanding of concepts such as regressions, dependent and independent variables to be able to understand this. This also applies to the understanding of a domain; it is required to have an understanding of the conceptual model of the domain to validate if certain questions or statements fit in that model.

To assess the knowledge that the participants have gained we will assess multiple levels of understanding. In practice, the text-based domain knowledge would be available, but in this case, they are being removed to be able to test the quality of the mental representation. Based on some studies in this area (Gemino and Wand 2005, Burton-Jones and Meso 2006), we will measure surface and deep understanding by using actual measures. Surface understanding will validate if certain knowledge fits in the conceptual model. Deep understanding is the ability to use the knowledge in the conceptual model to draw conclusions from it in a problem-solving context. Surface understanding will be measured by multiple-choice questions. According to Green multiple-choice or true/false items can address concept definitions, broad conclusions or particular steps in an argument (Greene, 2001). To measure deep understanding of the domain, problem-solving questions and a Cloze test can be used. The problem-solving questions will contain questions which are not directly solvable by discussed material but closely related to it. The Cloze test also provides the possibility to measure more subtle differences in understanding of the domain knowledge. Green wrote that the Cloze format can test a student's recognition of the cohesive devices that make a carefully constructed argument possible, just as writing an essay would (Greene, 2001).

3.2 Technical design: elaboration of the method

This experiment is designed as a joint effort between two graduates of the BPMIT program; Sebastián Groosman and Jos Huiting. While the design and execution of the experiment were performed together, both theses have a separate research question. This thesis focused on the first part of the experiment that was conducted to obtain domain knowledge from conceptual models.

The research question that needs to be answered during the experiment is: *Does the usage of conceptual models with domain knowledge lead to a better understanding of the subject domain?* An experiment was designed where participant gathers knowledge about a problem which has a substantial amount of domain content. To answers this research question, we've established a hypothesis that we will confirm or reject to determine the outcome of this research:

H_0 *There is a significant difference in domain understanding between the group that used the conceptual model and the group without one.*

Two groups of participants get a knowledge model and or a text that describes a problem domain. After obtaining the knowledge for a limited amount of time, a questionnaire is performed. To assess the knowledge that was gained and validate if there are significant differences in domain knowledge between the groups. This design was partly based on the research performed by (Gemino & Wand, 2005), although that research had a primary endpoint to compare entity-relationship model grammars.

Sample size
Due to the nature setup of the research, we will use non-probability sampling to determine the correct sample size. Given that the group is homogeneous and we leverage the minimal sample sizes provided by Saunders we need to add 4-12 participants per experiment (Saunders et al., 2008, p. 297). This setup allows for testing the research questions while avoiding learning effects. During the three experiments 24 participants are included.
The participants consisted of:

- Software-related professionals working at Mendix R&D, 8 participants (experiment in English).
- Student software engineers at Educom, 8 participants (experiment in Dutch).
- Software-related professionals working at Beter Bed, 8 participants (experiment in Dutch).

Depending on native language the the questionnaire was supplied in Dutch or English for these experiments. The attached research questions and answering model served as a basis for these. Participants had a varied background, in terms of job title, background and experience which is also part of the questionnaire. To gather more insight in their ability to model (domain) knowledge their generic knowledge about system modeling and more specifically about UML is self-assessed with a 7-point Likert scale as part of the questionnaire.

3.2.1 Experiment

Participants were randomly assigned into two groups. Groups were asked to sit on opposite sides of the room to avoid learning effects. The experiments were conducted in rooms where no other activities happened during the experiment. The participants aren't allowed to deliberate during the experiment.

Introduction (5 min)
During the introduction, the participants got the outline of the experiment, alongside a brief introduction about the problem domain. This introduction did not go into domain details on purpose, to avoid a big difference in information over the various iterations of the experiment. The same PowerPoint presentation with information was used in every iteration, just translated for the occasion.

Participants gain domain knowledge (15 min)
Participants get a detailed, the control group gets a textual domain description, which for the test group is accompanied by a knowledge model in UML. The problem is not limited to the work context, to avoid participants with too much specific knowledge about the subject which affect the outcomes. This distinguishes the groups that work with and without a model. In this first part of the experiment the participants study the information from the provided domain. They also get blank paper to enable them to model the domain or take notes. These notes and provided domain knowledge are collected before the questionnaire is conducted. This enables this experiment to assess the quality of the mental model that the participants have built around the domain. During the explanation of the experiment it was clearly stated that the answers to the questions should match the content of the domain model and that participants should stay within the subject domain during the questionnaire. A short summary of the domain description is provided below.

Domain description
Acme is a company that takes care of the maintenance of planes on Schiphol. The maintenance is performed by engineers in certain maintenance locations. Planes issue alerts that result in procedures and tasks that need to be performed by engineers. An application has been built that should help the planner manage the repair work, based on alerts and other input by engineers.

A mockup application with a few dashboard and overview pages is briefly shown to the participants during the introduction. The full text and knowledge model are available in *attachment 1 and 2*.

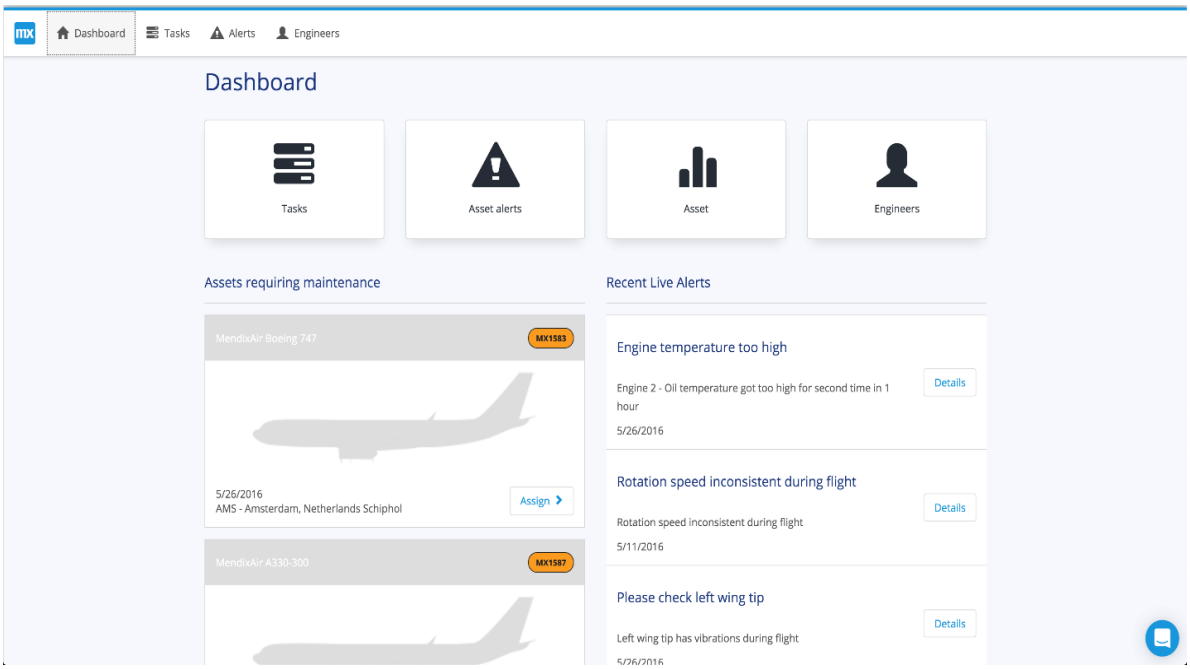


Figure 2: Mockup of the application used in the experiment

Participants fill questionnaire – (15 min)
With the questionnaire, we will measure the knowledge which the participants gained. To more precisely measure the knowledge that the participants have we will assess multiple levels of understanding. Based on the previous chapter we distinguish shallow and deep understanding. We will measure surface and deep understanding by using actual measures.

Surface understanding – Multiple-choice questions (10 questions)
As a first step in the questionnaire, participants answered ten questions around the domain that was provided. The goal of these questions was to measure how much shallow understanding of the domain was gathered amongst the participants. Each of the questions was marked to contain knowledge that was captured in a certain part of the model or text. Two of the questions did contain knowledge that was not specified in the domain description.

Example: Work can be performed in multiple maintenance locations during a maintenance procedure of an aircraft?

Anwer (Yes / No): Yes, there is only one active maintenance location that is attached but this one can change during the procedure. This is expressed in the model as well as the text.

Deep understanding – Problem solving (4 questions)
The first type of questions to measure deep understanding that we use are problem-solving questions. These questions are not directly solvable by discussed material but closely related to it. The answer will be checked against a pre-compiled list of acceptable answers.

An example of a problem-solving question: How does the maintenance process ensure the quality of the performed task? Please name the four implemented measures
One of the correct answers: The usage of standardized procedures and tasks

Deep understanding – Cloze test (2 tests, consisting of 10 + 4 questions)
The Cloze test also provides the possibility to measure more subtle differences in understanding of the domain knowledge. The answer will be checked against a pre-compiled list of acceptable answers.

Example Cloze test: Based on the availability of a maintenance location / hangar / location[C1.1] an asset / aircraft [C1.2] can be repaired by an engineer [C1.3].

A full overview of all questions that were used during the questionnaire is available in *attachment 2*.

At the end of the experiment, some additional data was gathered on how the participants experienced the difficulty of the experiment and results. This can be used for further analysis about the participants and might give additional insights on how to improve the experiment.

3.2.1 Scoring system

To validate the multiple-choice questions the answer is checked, the result of the answers leads to a percentage of correct answers. A score will be calculated for questions that are marked to contain knowledge that is in the Model-only [M], Model and Text [MT], Text-only [T] or Neither (N). Also, a total score overall multiple-choice questions will be calculated as a percentage.

The answers to the problem-solving questions will be checked against a pre-compiled list of answers. These are checked against the list twice, by two independent reviewers, in this case the authors. If there are still differences after two rounds the average value is used for the statistical analysis. To determine agreement between the two reviewers, a Pearson correlation coefficient between the coders will be calculated. For the problem-solving question the score will be calculated by getting the percentage of the number of acceptable answers divided by the amount of provided acceptable answers. For the Cloze test, the pre-compiled list contains all synonyms if the answers are corrected the value will be 1, otherwise it will be 0. For each of these types of questions, a total score will be calculated.

To get a total score for a candidate the percentage for every category (multiple-choice, problem-solving and Cloze test) will be equally weighed to calculate the total score.

$$S^{total} = (X^{Multiple\text{-}choice} + Y^{Problem\text{-}solving} + Z^{Cloze}) / 3$$

3.2.2 Primary and secondary endpoints

To answer the primary endpoint of this study, the overall scores for the test and control groups are preperare. The total score calculated is compared over these two groups, and determined if there are significant differences. Secondary, we explored the subgroups of questions to see if the amount of model-related experience skews the outcomes of the tests. As exploratory endpoints; we analyze subgroup based on modeling experience, work experience and other baseline characteristics.

3.3 Statistics

In our inclusion criteria, we described measures to include a sample size that allows to actually measure differences within the group. The participants are assigned randomly to a group at the beginning of the experiment.

This experiment will result in continuous data in two independent groups. Therefore, based on normality distribution, unpaired T-tests or Mann-Whitney U tests will be used to compare the groups. Results will be reported as means with standard deviations.

As described in paragraph 3.2, both authors score the outcomes of the questions measuring deep understanding individually. To determine the level of agreement between the ratings, Pearson's correlation is calculated. Pearson was used because the type of data was continous and if a linear relationship between the results exists.

Data gathered during this research will be stored anonymously. This data will be treated confidentially and stored in a secure way within Europe.

4. Results

The experiment has been performed in December 2019, at three locations in the Netherlands. This section described what changed compared to the planned and which data was gathered from the execution of the experiment.

4.1 Deviations from the plan

This paragraph describes the deviations from the plan compared to the method as it was described.

It was planned to split the groups in every test and let them work from separate rooms. However, for the first introduction, this is done together to make sure that both authors provided the exact same information (especially domain knowledge) to both groups. To make sure that everyone received the same information it was executed that way for all groups.

The experiment was performed in both English and Dutch. During the translation between English a Dutch an error was made, and the wrong question was used in the English experiment. Hence, the question (1.8 for participants in the test) was excluded from the experiment for everyone. A similar error was made in the Cloze test question C2, which led to a deviation in results between the three groups. Question C3 was removed from the results and mark the results of C2 for those participants as "not available". For question 1.10 there was a small variation between the questions between some groups that we first administrated separately. After reviewing the results, it appeared that there was not a big difference, hence we merged the results back for both questions.

A correction was also made to the pre-compiled set of answers after the test. For multiple-choice question 1.2 the category was changed to 'None', as it became apparent that it wasn't stated clearly in the text nor model.

4.2 Experiment - Participants

As part of the experiments, three groups of eight participants were included in the research. One of the participants didn't understand how to correctly fill complete the experiment and the results were therefore excluded from the experiment. The includes participants are randomly assigned to group A(n=11) of B(N=12). Group A consisted of participants that both used a model and text, while group B was the group that used only a text. The 23 included participants successfully completed the tests.

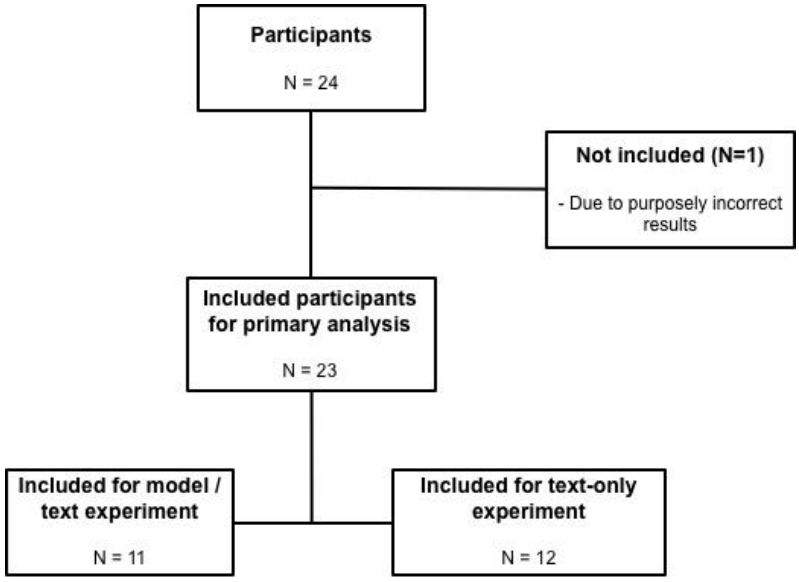


Figure 3: Inclusion criteria of the participants

To give an overview of the participants and their characteristics a baseline table in Figure X was included. These characteristics were gathered in participant information tests at the start of the questionnaire.

	Model and text (N (%) or mean (SD))	**Text only (N (%) or mean (SD))**	**P-value**
Number of participants	11 (48%)	12 (52%)	
Educational level			0.534
University	8 (73%)	8 (67%)	
HBO	2 (18%)	3 (25%)	
MBO	0 (0%)	1 (8%)	
VMBO	1 (9%)	0 (0%)	
System modeling knowledge (scale 0-7)	3.8 (1.4)	4.5 (0.9)	0.177
UML knowledge (scale 0-7)	3.3 (1.6)	3.4 (1.1)	0.803
Working experience (years)	5.2 (7.0)	7.5 (6.2)	0.41

Figure 4: Baseline table participants

4.3 Experiment - Results

4.3.1 Multiple choice questions

To make sure a meaningful set of multiple-choice questions was used these are categorized to indicate which purpose they serve and what they measure. Based on the experiment the questions are categorized if the knowledge is in the knowledge model, text, both model and text or none. The last category of questions serves as a control question to indicate if the participants have an understanding of the domain.

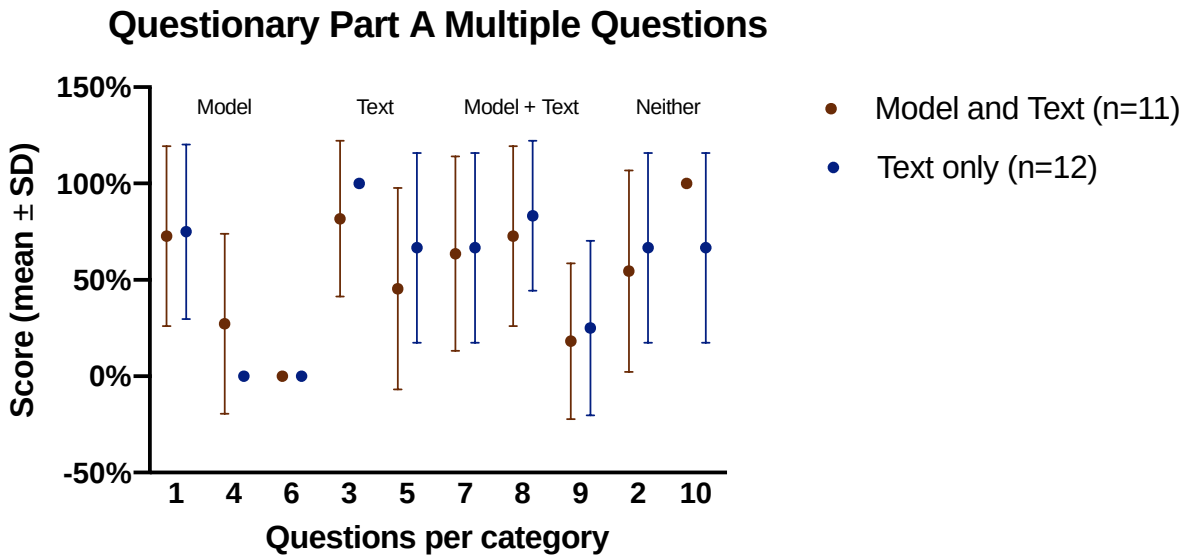


Figure 5: Visualization of multiple-choice categories

The scores for the multiple-choice total 55.00% and 53.64% in respective groups A and B (p=0,792, figure 6). The standard deviation was 13.14 and 11.20 for groups A and B, with a range of 40,00 for both groups.

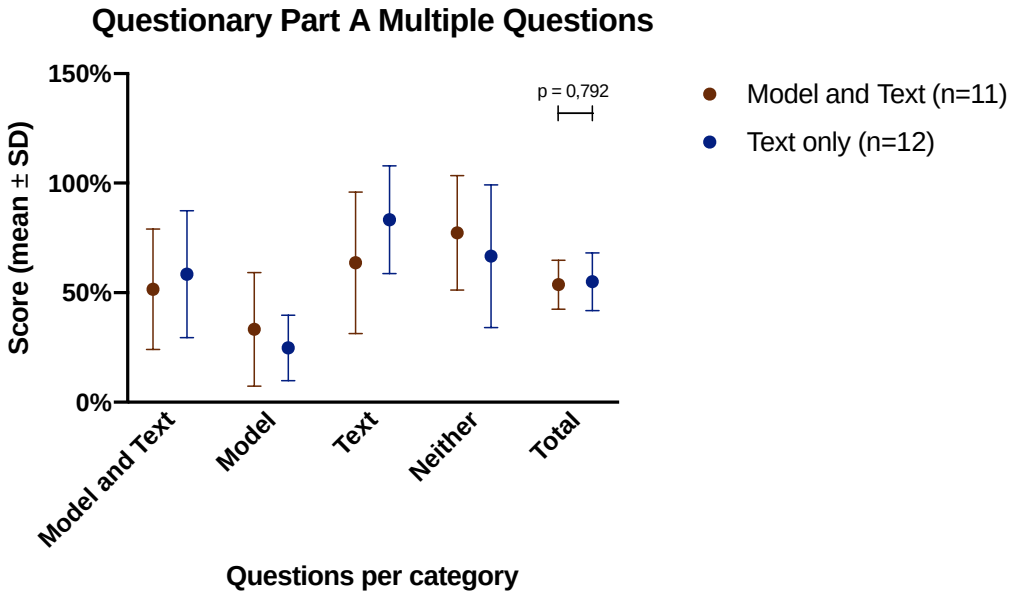


Figure 6: Visualization of multiple-choice question results

4.3.2 Problem solving questions

The scores for the problem-solving questions total 50.58% and 45.74% in respective groups A and B (p=0,297, figure 7). The standard deviation was 10.59 and 11.39 for groups A and B.

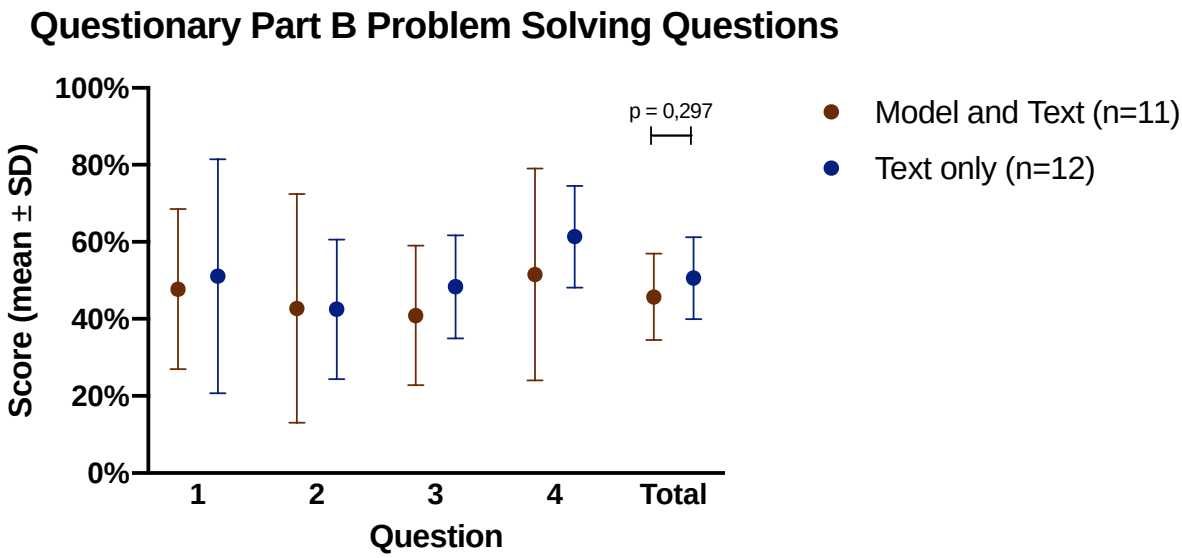


Figure 7: Visualization of problem-solving question results

4.3.3 Cloze test

The scores for the first Cloze test total 60.00% and 69,55% in respective groups A and B (p=0,088, figure 8). The standard deviation was 12.79 and 12.74 for groups A and B.

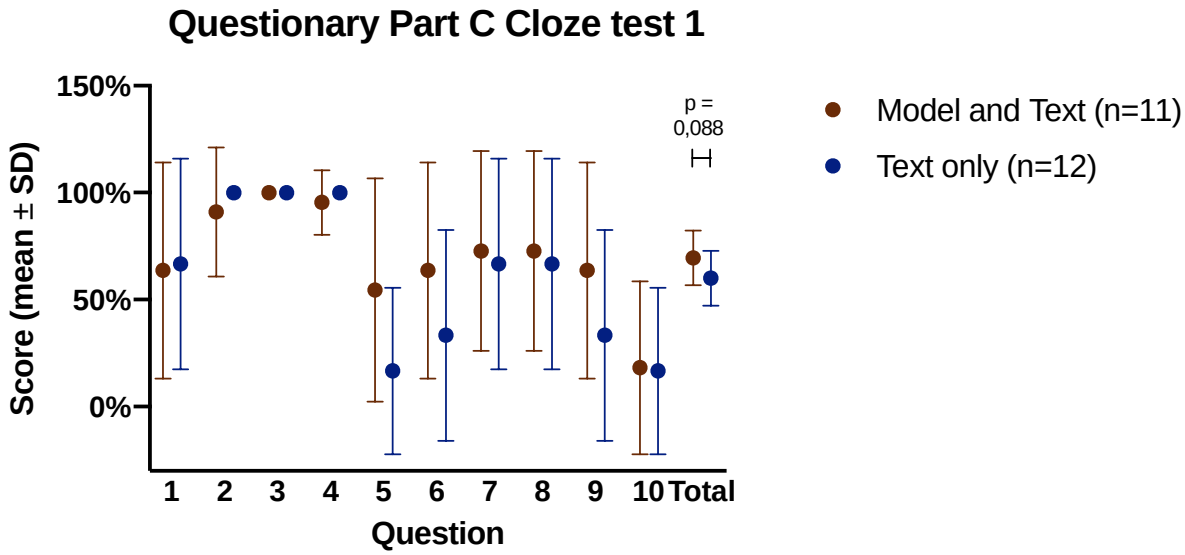


Figure 8: Visualization of Cloze test results part 1

The scores for the second Cloze test total 84.00% and 78,13% in respective groups A and B (p=0,552, figure 9). The standard deviation was 21.22 and 16.02 for groups A and B.

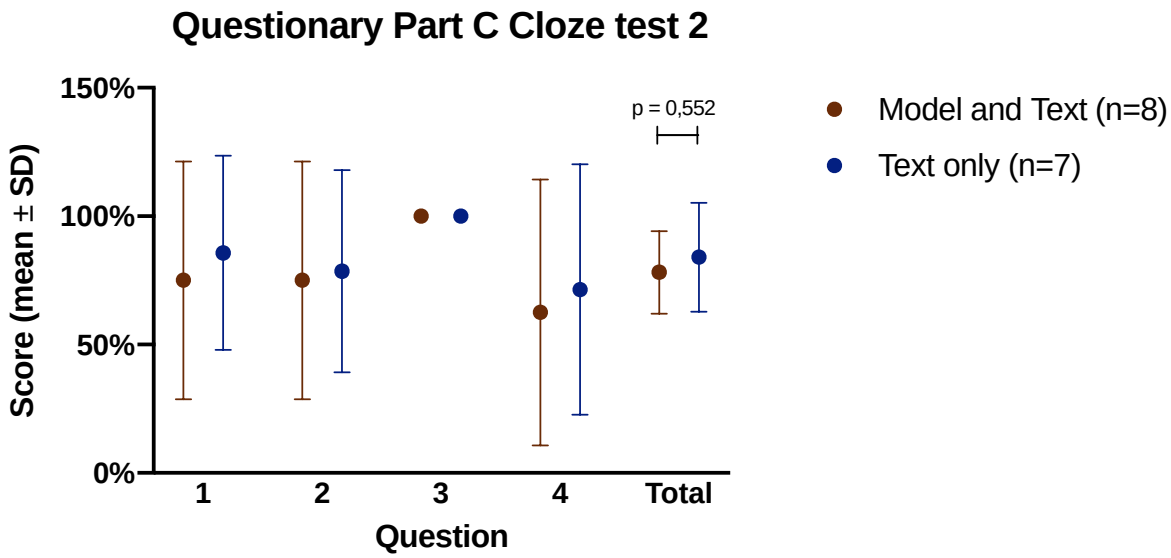


Figure 9: Visualization of Cloze test results part 2

4.3.4 Score model

Scored amount of acceptable per question, as the procedure described in 3.5.2. The agreement of the two coders was checked using a Pearson, which was 0,974. This indicates a high level of agreement between the coders; hence the results are closely matched. For the results that didn't match the average score was used but given the high degree of impact, this is not likely to have a significant impact on the outcome of the results.

4.3.5 Overall results

The scores for the combined tests total 57.42% and 57,00% in respective groups A and B (p=0,882, figure 10). The standard deviation was 7.78 and 5,07 for groups A and B.

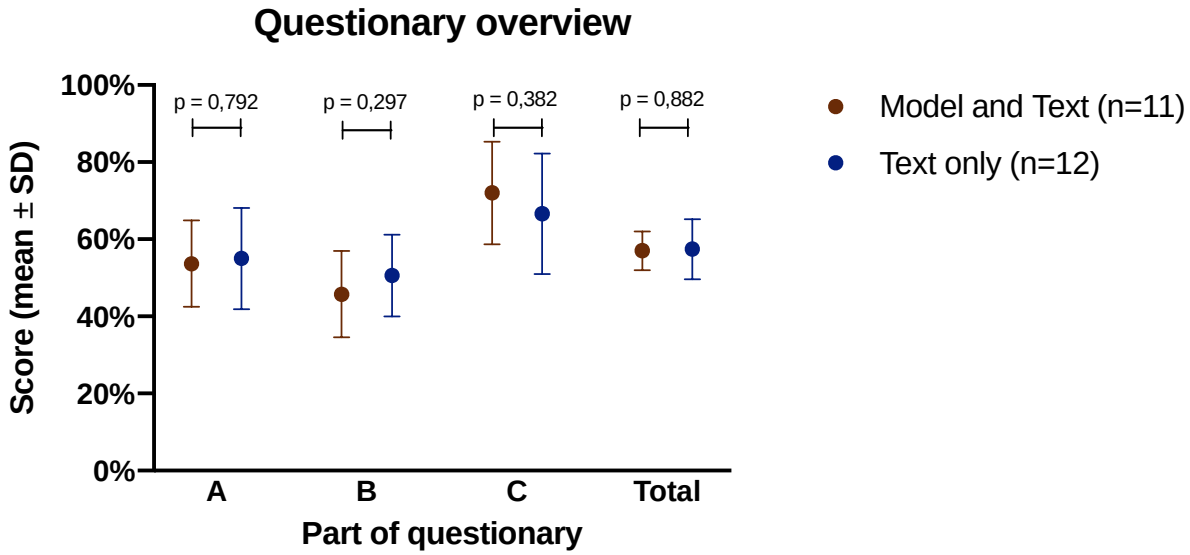


Figure 10: Visualization of overall results

4.3.6 Exploratory results - System modeling experience

To see if the system modeling experience played a big role the groups were divided differently based on their amount of system modeling experience (opposed to text / model and text). This yielded quite similar results as the full scores for the questionnaire overview. The scores for the combined

tests total 56.25% and 57,30% in respective groups A and B (figure 11). The standard deviation was 6,61 and 4,78 for groups A and B. Given this was just an exploratory test with a similar outcome to the previous results the unpaired t-test was omitted.

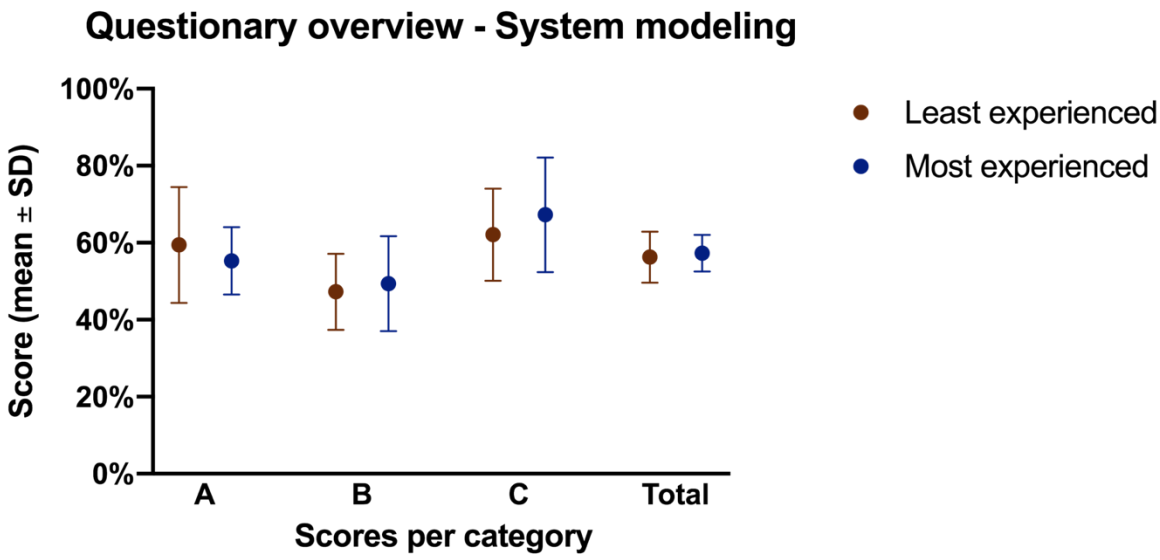


Figure 11: Visualization of explority results, system modeling

5 Discussion

5.1 Discussion - Participants

To ensure that both groups of participants are comparable, a baseline table was added to compare the characteristics of the participants included to the test. This baseline table contains basic data about the groups, with the results of an unpaired t-test to indicate the significance of the results. Group A is the group that has both a model and a text, while group B only used a text to gain domain knowledge.

The educational background of the participants mostly consisted of universities, while the remaining six candidates noted Higher Vocational Education or Vocational Education. Due to the fact that the group was partly international, it is difficult to compare this between countries. Some universities in e.g. the US are comparable to the Higher Vocational Education in the Netherlands. Based on these numbers we assume that there is a reasonable amount of theoretical knowledge about the subject matters as part of their studies. Between groups, there is no noticeable nor significant difference amongst the groups.

A bigger difference was noticed in the system modeling skills. For group A this averaged at 3.8 while group B scored 4.5. For UML the score for group A was 3.3 while for group B it was 3.4.
We can partly relate these differences to a difference in work experience. For the group A the average work experience in years is 5.2, while for group B it is 7.5. This might explain why the text-only group rates the system modeling knowledge higher than the other group as they had more time to develop these skills as a practioner. However, the same doesn't apply for UML, both groups have similar results for that category. This might be explained if most of the knowledge gained around UML is theoretical and isn't extended much by applying it in practice.

Overall the comparison of the groups did not indicate significant differences between participants, which is summarized in the baseline table results in paragraph 4.2

5.2 Discussion - Results

Based on the difference in materials that are supplied (just a text or both model and text) we expected a significant difference in understanding of the domain model. In the research setup, we discussed how we measure for surface and deep understanding of the domain using different kinds of questions.

Surface understanding

For surface understanding, we expect a more similar outcome for both groups, as most of these facts are more easily deductible from the material that is provided. For certain questions that contain knowledge only provided in the model, we expected a difference.

Multiple-choice questions

For this group of questions, the results are categorized in Model, Model & Text, Text and None as detailed in paragraph 4.3.1. For the category Model group, A scores higher on the model-related questions, but the difference is quite small. Surprisingly all participants failed to answers Q6 correctly, which leads to the question if that knowledge was expressed clearly in the model. Or it indicates that the participants did not use the model actively. The group that focusses just on the text scores better on both the text-only as well as the questions that are both in model and text. Especially the latter is unexpected, as people with both the model and text should be able to deduct this question more easily. This also raises the questions if there were able to do so?

Overall there was no significant difference between group A and B, with scores totaling 55.00% and 53.64% in respective groups A and B

Deep understanding

For measuring deep understanding we expected a significant difference between the groups as solving these questions requires a deeper understanding of the domain. Based on our theoretical framework we expected that the transfer of domain knowledge would be improved by the usage of the conceptual model.

Problem-solving questions

The scores for the problem-solving questions total 50.58% and 45.74% in respective groups A and B. From the results it seemed that the model-group had quite a big range in the average score of the questions, suggesting that the members of the group varied largely in the answers they gave.

The average in group A is just slightly, but not significantly, higher compared to group B.

Cloze tests

For the first Cloze tests, the results were 60.00% and 69,55% and for the second 84.00% and 78,13% in respective groups A and B. Given this test mimicked writing an essay we did expect the biggest difference between the groups. So, the results for the first test was most surprising, as the group without the model scored higher.

For both outcomes there was no significant difference in results between group A and B.

Overall score

To get an overview of the total results we took the results of the previous categories and weighed them equally. The scores total 57.42% and 57,00% in respective groups A and B. The total results are really close, so this clearly indicates that there are no significant differences in results between group A and B overall for this test. This was confirmed by the T-test that was applied.

Additional data was gathered by splitting the groups based on their self-assessed score in system modeling experience. One group that had the lowest scores, and the other groups with the highest scores. This yielded quite similar results as the groups split by model and text. It seems however that this group scored lower for multiple-choice questions but slightly higher for problem solving (although far from any significant result).

Based on the literature it became clear that the gap in domain knowledge is a big threat to the success of successful software projects. Based on the reduction in the size of the knowledge gap this may lead to better team results, maybe even better information architectures. Shared understanding in a team might help for that, which can be carried by conceptual models. With the lack of significant differences between the groups the questions rise how a conceptual model can contribute to knowledge transfers. Although this experiment should be further validated, the results indicate that the knowledge transfer is not improved by having a model in this experiment. One of the questions that rise is if the process of making the model is more important than having one? If both groups had drawn their diagrams instead of receiving them in the experiments, this might have affected the results. This also might have occurred within the experiment as participants from the text-only control group stated modeling on paper.

Based on the literature we can still recommend teams to create conceptual models, as the lack of domain knowledge is a big risk for project failure. Aside from the possible benefits for software creation there might be other benefits of having domain knowledge documented in conceptual models. This might be beneficial for the maintenance of software, or the onboarding of new teammates. However, this can only be useful if the conceptual models are up to date with the reality of the software system.

5.3 Discussion – Limitations & reflection

The cohort of participants in this research are (mostly) employed the same company of both researchers that designed the experiment. Participation in this experiment was voluntary for all participants. They are informed upfront that they are free to withdraw from the study at any time.

Due to the short timeframe to set up this experiment only a minimal amount of experiments was conducted. The experiment was performed once with one participant as a try-out, before starting with the groups that are included in the experiment results. If time permitted this experiment should be performed more often, which allows for a better setup which captured the results in a more optimized way. Despite these limitations the location,

One factor that also might affect the reliability of this research is the fact that the research was conducted in Dutch as well as in English. One group of participants (at Mendix) consistent mostly from expat workers, which required a translated model, presentation and domain text. During the translation of the model we tried to keep the translation as close as possible, but this wasn't validated by a scientifically sound method.

Before the experiment, the knowledge of the questions was validated by the authors in the test, and for multiple-choice questions, it was categorized what it tested. During the experiment we noticed that quite a few participants questioned the answers we supplied in the discussion about part 1. For the problem-solving questions, we also did not relate the answers to the questions to the part of the text or model where it was supplied. With the current research method a discussion about the answers is subjective, as you can't prove if certain knowledge is expressed in the model. A UML knowledge model can be used to instantiate pieces of information, but it remains incomplete and is potentially not fully consistent with the supplied text. A formal notation could have helped here as it allows for mathematical validation of the result.

During the experiment, we also observed that quite a few participants in the text-only group started modeling themselves on paper. This might also affect the results, given that effectively also had a knowledge model based on their understanding. Effectively these participants both also had a model and a text, so that gives the other groups less of an advantage. If we assume that *making* a model is more effective than *having* a model these models might be more effective as well. That leads to the question if the experiment should be set up differently, to verify the effectiveness of conceptual models.

Another question about the experiment is if this is the right way to validate the knowledge that participants gained about a domain. Or is it too much geared towards testing reading abilities and knowledge comprehension for the participants in general? This might have been avoided with a different setup of the experiment, where reading is less important. The setup of this design was based on retention as described by Geminio which was used in a comparable experiment (Gemino & Wand, 2005).

Below we summarize the main measures that are previously discussed which ensured that this research was conducted reliably and is valid:

- Deliberate choice for the search method that is being used, explained in paragraph 3.1.
- A subject was prepared that did not relate to the domain knowledge of the participants
- Design of the research was thoroughly prepared, tested on one test subject and executed three times with the exact same materials.
- Established relation between multiple-choice questions to the content they represent from the supplied domain (e.g. categories in multiple-choice questions)
- Answers to the problem-solving questions are rated by two people, and the results compared statically.
- Random sampling with a group with minimal size, validated by usage of baseline table to compare groups.
- Avoid learning effects by moving participants to different parts of the room and lot allowing discussing the contents.
- Usage of statistics in results to discuss the significance of differences

5.4 Conclusions

For the experiment we didn't note any significant differences between the groups that participated in the experiment. There was however a difference in work experience and modeling knowledge that might have affected the results. For every type of questions, the results of the groups were quite comparable, and overall there was no significant difference between them.

H_0 There is a significant difference in domain understanding between the group that used the conceptual model and the group without one.

Result The hypothesis is rejected, there is no significant difference for the group that used a conceptual model and the group without one.

Similar results were yielded when the group was split on properties of the participants, such as system modeling experience. This yielded quite similar results as the groups splitted by model and text.

5.5 Recommendations for practice

During the literature research of this thesis, it became clear that the transition to agile was quite a shift in the way information modeling was performed. The information modeling was suddenly not an embedded part of the process anymore. However, as it turns out that the shared understanding of the (agile) team is crucial to successfully deliver software. The gap in domain knowledge between the business and the team is a key factor for failure in projects. Some researchers even think that fixing this gap is crucial for the survival of organizations.

One recommendation is to experiment with various models to transfer domain knowledge and increase conceptual knowledge in a team. This could take the shape of the conceptual model but could also be documentation or have conceptual discussions with stakeholders more often. Try to find a way that helps to tighten this knowledge gap, on both the conceptual level as well as the technical level. There are also attempts to bring this domain knowledge to the code of the teams, by e.g. implementation of domain specific languages for end-users or the application of domain driven design in the codebase.

5.6 Recommendations for further research

During the literature research and experiment the question rose what kind of knowledge actually should be captured in a knowledge model. Are there different types of knowledge models that could be leveraged here? And what knowledge is required to close the knowledge gap and make the project more successful.

One limitation of the current setup of the experiment with UML models was the capability of the models to express knowledge. The knowledge models that are used contained basic capability to express knowledge as entities and relationships. However, compared to more formal approaches, that e.g. use relational algebra the amount of certainness that knowledge actually is expressed correctly in the model is limited. When a method such as Ampersand (Michels et al., 2011) is used it can be mathematically proven that certain knowledge is expressed correctly in the model, hence that questions and the knowledge model are better aligned.

References

Abdullah, N. N. B., Honiden, S., Sharp, H., Nuseibeh, B., & Notkin, D. (2011). Communication Patterns of Agile Requirements Engineering. In *Proceedings of the 1st Workshop on Agile Requirements Engineering* (pp. 1:1--1:4). New York, NY, USA: ACM. https://doi.org/10.1145/2068783.2068784

Abrahamsson, P., Warsta, J., Siponen, M. T., & Ronkainen, J. (2003). New directions on agile methods: a comparative analysis. In *25th International Conference on Software Engineering, 2003. Proceedings.* (pp. 244–254). https://doi.org/10.1109/ICSE.2003.1201204

Al-Salti, Z., & Hackney, R. (2011). Factors impacting knowledge transfer success in information systems outsourcing. *Journal of Enterprise Information Management*. https://doi.org/10.1108/17410391111166521

Ambos, T. C., Ambos, B., Eich, K. J., & Puck, J. (2016). Imbalance and Isolation: How Team Configurations Affect Global Knowledge Sharing. *Journal of International Management*. https://doi.org/10.1016/j.intman.2016.03.005

Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., … Thomas, D. (2001). Manifesto for Agile Software Development. *Manifesto for Agile Software Development*. Retrieved from http://www.agilemanifesto.org/

Cockburn, A., & Highsmith, J. (2001). Agile software development: The people factor. *Computer*. https://doi.org/10.1109/2.963450

Davis, A. M. (1988). A taxonomy for the early stages of the software development life cycle. *Journal of Systems and Software*, *8*(4), 297–311. https://doi.org/https://doi.org/10.1016/0164-1212(88)90013-1

Dybå, T., & Dingsøyr, T. (2008). Empirical studies of agile software development: A systematic review. *Information and Software Technology*, *50*(9), 833–859. https://doi.org/https://doi.org/10.1016/j.infsof.2008.01.006

Evans, E. (2004). *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional.

Fowler, M. (2011). CQRS. Retrieved July 15, 2019, from https://martinfowler.com/bliki/CQRS.html

Gemino, A., & Wand, Y. (2005). Complexity and clarity in conceptual modeling: Comparison of mandatory and optional properties. In *Data and Knowledge Engineering*. https://doi.org/10.1016/j.datak.2004.12.009

Glinz, M., & Fricker, S. A. (2015). On shared understanding in software engineering: an essay. *Computer Science - Research and Development*. https://doi.org/10.1007/s00450-014-0256-x

Greene, B. (2001). Testing reading comprehension of theoretical discourse with cloze. *Journal of Research in Reading*. https://doi.org/10.1111/1467-9817.00134

Haldin-Herrgard, T. (2000). Difficulties in diffusion of tacit knowledge in organizations. *Journal of Intellectual Capital*. https://doi.org/10.1108/14691930010359252

Hoffmann, A., Bittner, E. A. C., & Leimeister, J. M. (2013). The emergence of mutual and shared understanding in the system development process. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-642-37422-7_13

Ichijo, K., & Kohlbacher, F. (2008). Tapping tacit local knowledge in emerging markets - The Toyota way. *Knowledge Management Research and Practice*. https://doi.org/10.1057/kmrp.2008.8

Michels, G., Joosten, S., Van Der Woude, J., & Joosten, S. (2011). Ampersand: Applying relation algebra in practice. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. https://doi.org/10.1007/978-3-642-21070-9_21

Mylopoulos, J. (1992). Conceptual Modelling and Telos 1. In *Conceptual Modelling, Databases, and CASE: an Integrated Vie of Informaion System Development*.

Nonaka, I., & Takeuchi, H. (1995). The knowledge creating company. *Harvard Business Review*. https://doi.org/10.1016/0024-6301(96)81509-3

Nuseibeh, B., & Easterbrook, S. (2000). Requirements Engineering: A Roadmap. In *Proceedings of the Conference on The Future of Software Engineering* (pp. 35–46). New York, NY, USA: ACM. https://doi.org/10.1145/336512.336523

Olmos, K., & Rodas, J. (2014). KMoS-RE: Knowledge Management on a Strategy to Requirements Engineering. *Requir. Eng.*, *19*(4), 421–440. https://doi.org/10.1007/s00766-013-0178-3

Partridge, C., Gonzalez-Perez, C., & Henderson-Sellers, B. (2013). Are Conceptual Models Concept Models? https://doi.org/10.1007/978-3-642-41924-9_9

Rubin, E., & Rubin, H. (2011). Supporting agile software development through active documentation. *Requirements Engineering*, *16*(2), 117–132. https://doi.org/10.1007/s00766-010-0113-9

Saunders, M., Lewis, P., & Thornhill, A. (2008). *Research Methods for Business Students 5th Ed*. *Research methods for business students*. https://doi.org/10.1007/s13398-014-0173-7.2

Schön, E.-M., Thomaschewski, J., & Escalona, M. J. (2017). Agile Requirements Engineering: A systematic literature review. *Computer Standards & Interfaces*, *49*, 79–91. https://doi.org/10.1016/J.CSI.2016.08.011

Spano, R. (2005). Potential sources of observer bias in police observational data. *Social Science Research*. https://doi.org/10.1016/j.ssresearch.2004.05.003

Tan, M. (1993). Establishing mutual understanding in systems design: An empirical study. *Journal of Management Information Systems*.

van Lamsweerde, A. (2002). Requirements engineering in the year 00: a research perspective. https://doi.org/10.1109/icse.2000.870392

Wang, E. T. G., Chia-Lin Lin, C., Jiang, J. J., & Klein, G. (2007). Improving enterprise resource planning (ERP) fit to organizational process through knowledge transfer. *International Journal of Information Management*. https://doi.org/10.1016/j.ijinfomgt.2007.02.002

Attachment 1 – Experiment text

Airplane Controlled Maintenance Enterprise (Acme) is a company which is primarily focused on the maintenance of aircrafts. It is located at Schiphol, where they have multiple hangars from which they performance maintenance. From every hangar only one maintenance action can be performed at the same time. Every airplane that returns to its base station Schiphol will deliver input to the maintenance system.

To easily keep track of the current state of maintenance activities a mockup created of an app that helps managing this. It contains views with information on tasks performed by engineers, alerts, engineers and the aircrafts. The main dashboard shows an overview of items of the highest importance. It contains a list of recent malfunctions, with a date and some short descriptions. These malfunctions are based on the alerts that are stored by the planes during flight; these are automatically uploaded when the plane lands at Schiphol. Based on the severity of the alert the maintenance will be planned with a certain urgency, the planner will take care about handling this and ensures that a procedure is started. This procedure consists of multiple tasks, assigned to an engineer.

On the dashboard there is also an overview of all assets that require maintenance. This overview contains the plane's manufacturer, type, planned maintenance and its current location. The regularly scheduled maintenance depends on a couple of variables, such as flight hours, age and type of the plane. This overview can be used to assign maintenance tasks to an engineer.

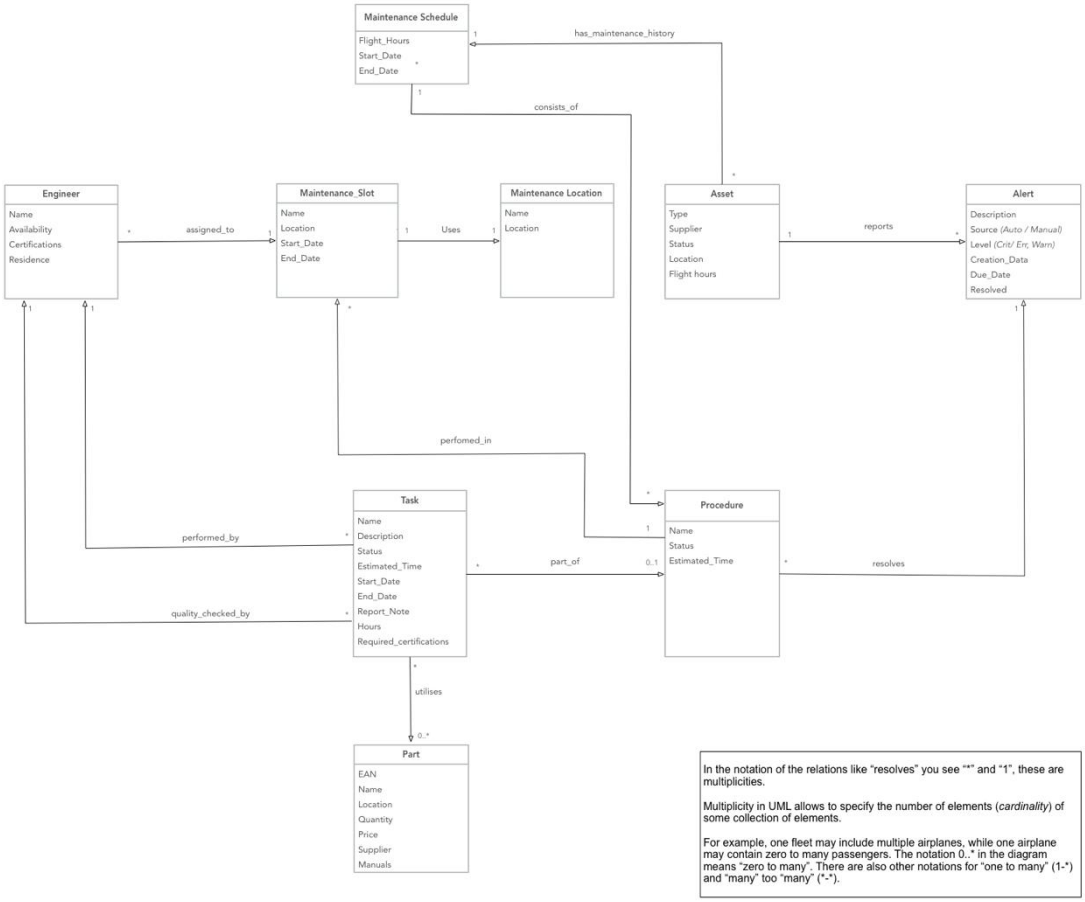
One of the top priorities in aviation industry is security. This results in a lot of rules and regulation concerning maintenance and thus results in standardized procedures and tasks for all kinds of maintenance. As part of a task, parts can be replaced, for each of which a specific manual should be available. To enable the planner to plan ahead, every task and procedure have an estimation of the time required to perform it.

Engineers work according to standardized procedures, for both the regular maintenance work as well as for resolving malfunctions indicated by alerts. After all procedures and tasks are finalized an engineer should mark an alert as resolved. An engineer should be qualified to perform certain tasks, which is regulated by certification. Only certified engineers can be planned to perform tasks that have requirements on this. An engineer should report on working hours and note down his remarks. This is kept in the maintenance logs of the asset; all of the performed procedures form its maintenance history.

An aircraft remains grounded as long as there are procedures being performed, or alerts haven't been resolved yet. A plane can be parked in the hangar, but only one plane at the time is under maintenance. If a more complicated procedure needs to be performed a plane can be moved to a different hangar, as some of the required equipment is not available in every maintenance location. All alerts have to be resolved by an engineer before the procedure is resolved. An alert can be marked as resolved by performing a procedure that resolves the issue right away or based on testing in the procedure decide to plan the maintenance later as part of the regular schedule. Based on this information the planner decides to put the airplane in or to take it out of operation. The hangar is available again for another maintenance procedure as soon as this has been done. If a plane reaches the maximum number of flight hours, it will be taken out of operation by the planner.

For this can you can assume that there's an unlimited amount of parts available in the local warehouse, which can be used without considering costs or whether it is in stock. The planner will make sure that the required materials and engineers will be in the right hangar on time.

Attachment 2 - Experiment model



Attachment 3 - Experiment questions and answers

Comprehension question (10x)

1. Maintenance tasks on an aircraft are performed by certified engineers, can an engineer be assigned to multiple maintenance locations per timeslot? **[Model]**
   No, see relation assigned_to from Engineer to Maintenance_Location in the knowledge model.

2. If an aircraft touches down on Schiphol and sends over its notice to the maintenance system it will always result in starting a procedure. **[Not in model or text]**
   Unknown, not in model or text.

3. The parts that are required for a maintenance task are determined while performing the task. **[Text]**
   No, tasks and parts are performed based on guidelines from a manual.

4. There are always multiple engineers involved for performing a maintenance task. **[Model]**
   Yes, see relation quality_check from Task -> Engineer in the knowledge model.

5. An alert with a status higher then warning leads to a single procedure, which consists of multiple tasks. **[Text]**
   No, an alert can consist of multiple procedures. See multiplicity Alert -> Procedure in the knowledge model.

6. All alerts are inserted just automatically to the maintenance system **[Model]**
   No, there's also a manual flow which facilitates pilots to pass information about a malfunction. See Alert – Source in knowledge model

7. The estimated time to perform a difficult procedure is always known before start. **[Model and Text]**
   Yes, check the Procedure's – Estimated_Time property in knowledge model and explanation in text.

8. Work can be performed in multiple maintenance locations during a maintenance procedure of an aircraft. **[Model and Text]**
   Yes, there's only one active maintenance location which is attached but this one can change during the procedure. This is expressed in the model as well as the text.

9. The only way to resolve an alert is to complete the procedure attached to it. **[Model and Text]**
   Yes, see relation "resolves" from Procedure -> Alert and explanation in text.

10. The planner contacts the pilot after touchdown to get the asset to the right location **[Not in model or text]**
    No, not in the model or text.

Problem solving (4x)

How does the maintenance process ensure the quality of the performed task? Please name the four implemented measures:

- *The usage of standardized procedures and tasks*
- *Quality check by another engineer (see quality_checked_by from Task -> Engineer)*
- *Certifications of engineers (see certification of Engineer)*
- *Parts are applied in a standardized way*
- *Plane can't leave with an error*

What should be taken into account while planning the work of an engineer?

- *The engineer should be available*
- *The engineer should be certified to perform a certain task*
- *The work location and / or equipment should be available, otherwise the assigned engineer can't work there.*
- *Urgency of the repair / alert*
- *Estimated time of the repair*

Which properties of an aircraft effect the maintenance schedule?

- *Age*
- *Amount of flying hours*
- *Type of aircraft*
- *Information from previous tests, procedures or other malfunction reports*
    - *Previous alerts, errors or failures*
    - *When was the last scheduled maintenance performed*
- *Current alerts, errors*

Which circumstances can keep an aircraft grounded?

- *Maintenance of an aircraft which is part of the regular maintenance schedule but isn't performed in time*
- *The plane has reached the maximum amount of flying hours*
- *Alerts / tasks are not resolved, the procedures of these are still running*

Cloze test (2x)

Based on the availability of a maintenance location / hangar / location C1.1 an asset / aircraft C1.2 can be repaired by an engineer C1.3. The performed procedure / repair / (maintenance) task C1.4 is based on an automatic / manually C1.5 or automatically / manually C1.6 inserted alert. A (maintenance) procedure C1.7 contains multiple tasks, within a (maintenance) task C1.8 multiple parts / materials C1.9 can be used. Every part / material C1.10 has a manual.

To release the asset / aircraft C2.1 a quality check / check / inspection / test /signoff C2.2 should be performed by another engineer C2.3 which ensures the plane can be set in operation / valid to fly / rotation / active C2.4 again.