



Improving Model Reviewing and Experimentation with Tool Support: A controlled experiment

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Dominik Kretz, BSc

Matrikelnummer 01529213

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.Ass. Dipl.-Ing. Dr.techn. Dietmar Winkler

Mitwirkung: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl

Wien, 10. September 2021

Dominik Kretz

Dietmar Winkler



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.



Improving Model Reviewing and Experimentation with Tool Support: A controlled experiment

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering & Internet Computing

by

Dominik Kretz, BSc

Registration Number 01529213

to the Faculty of Informatics

at the Technical University Vienna

Advisor: Univ.Ass. Dipl.-Ing. Dr.techn. Dietmar Winkler

Assistance: Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffel

Vienna, 10th September, 2021

Dominik Kretz

Dietmar Winkler



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Dominik Kretz, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 10. September 2021

Dominik Kretz



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Ein großer Dank gilt an dieser Stelle all jenen, welche mich während meines Studiums unterstützt haben. Allen voran möchte ich mich bei meiner Mutter und meinem 2019 verstorbenen Vater für die große Unterstützung in dieser Phase meines Lebens bedanken.

Ein weiterer Dank gilt Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffi und Univ.Ass. Dipl.-Ing. Dr.techn. Dietmar Winkler für die hervorragende Betreuung meiner Abschlussarbeit.

Wichtiger Bestandteil meines Alltags an der Technischen Universität Wien waren auch einige Studierende, welche ich während des Studiums kennenlernen durfte und, welche schnell zu guten Freunden wurden. Ein besonderer Dank gilt Christian Engelbrecht, BSc, Dipl.-Ing. Alexander Prock, BSc, Christoph Burger, BSc und Mustafa Isikoglu, BSc für die gute Zusammenarbeit und gegenseitige Motivation in den unzähligen Arbeitsstunden während unseres Studiums.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

At this point, I want to thank everyone who has supported me during my study. Especially, I thank my mother and my deceased father who passed away in 2019 for the big support during this part of my life.

Furthermore, I wish to thank Ao.Univ.Prof. Dipl.-Ing. Mag.rer.soc.oec. Dr.techn. Stefan Biffl and Univ.Ass. Dipl.-Ing. Dr.techn. Dietmar Winkler for the excellent supervision of my master thesis.

Another important part of my everyday life at the Technical University Vienna were different students that I got to know at the University and that quickly became good friends. Therefore, I want to especially thank Christian Engelbrecht, BSc, Dipl.-Ing. Alexander Prock, BSc, Christoph Burger, BSc und Mustafa Isikoglu, BSc for the good cooperation and the mutual motivation in the countless working hours during our study.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Der Prozess der Softwareentwicklung besteht aus mehreren verschiedenen Phasen. Um das Problem von Fehlern in Softwaresystemen zu lösen, wird die Software-Qualitätssicherung (SQS) angewandt. Um Fehler zu finden, werden in der Entwurfsphase des Softwaresystems Modelle als Artefakte verwendet. Diese Modelle sind eine vereinfachte Darstellung des realen Softwaresystems und können mit Hilfe von Modell-Reviews untersucht werden. Für den Entwurf und die Überprüfung verschiedener Modelle gibt es jedoch nur begrenzte Software-Unterstützung.

Häufig werden einzelne Review-Typen, Software-Unterstützungen oder allgemeine Methoden innerhalb einer wissenschaftlichen Studie verglichen. Die Verwaltung von Reviews und Experimenten ist jedoch eine ressourcenintensive Aufgabe. Daher ist es unser Ziel die derzeit begrenzte Software-Unterstützung für Modell-Reviews und die Verwaltung von Reviews und Experimenten weiterzuentwickeln.

Wir verwenden den *Model Design and Review Editor* (MDRE) und erweitern diesen mit zwei Prototyp-Komponenten. Die erste Komponente bietet Software-Unterstützung für die Meldung von Fehlern innerhalb von Modell-Reviews. Dadurch wird die Durchführung des Reviews verbessert und der Aufwand für die Datenerfassung reduziert. Zusätzlich implementiert die zweite Komponente eine Software-Unterstützung für die Verwaltung von Reviews und Experimenten. Das Hauptziel dieser zweiten Komponente ist die Bereitstellung von Aufgaben und die Überwachung des Fortschritts während des Experiments, um den Verwaltungsaufwand zu reduzieren. Zur Evaluierung unserer Ergebnisse führen wir ein kontrolliertes Experiment mit ca. 80 Teilnehmern durch, in dem wir die MDRE-Toolunterstützung mit dem traditionellen Ansatz vergleichen und die Ergebnisse analysieren.

Die Auswertung zeigt, dass die in den MDRE implementierten Erweiterungen in allen gemessenen Kategorien ähnliche Ergebnisse liefern wie der herkömmliche Ansatz, sowohl bei Modell-Reviews als auch bei der Verwaltung von Experimenten und Reviews.

Der MDRE ist also in der Lage, Reviews durchzuführen, die mit dem traditionellen Review-Ansatz vergleichbar sind. Darüber hinaus reduziert die implementierte Software-Unterstützung den Aufwand und vereinfacht den Prozess. Daher hilft der MDRE, Review und Experiment Managern die Planung, Durchführung und Verwaltung von Reviews und Experimenten mit weniger Ressourcenaufwand durchzuführen.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

In modern software engineering, the process of software development consists of multiple different stages. To address the problem of defects in software systems, Software Quality Assurance (SQA), a structured approach that aims to prevent and detect defects in software, was defined. To find defects, models are used as artifacts during the design stage. These models are a simplified view of the real software system and can be inspected using model reviews to identify defects within them. However, for the design and review of different models, there is only limited tool support available.

It is often important to compare individual review types, tool support components, or general methods within a scientific study like an experiment. However, managing reviews and experiments is a laborious task. Therefore, in this work, we aim to advance the currently limited tool support for model reviews and the administration of reviews and experiments. We use the *Model Design and Review Editor* (MDRE) in its initial state and aim to extend it with two tool support prototypes. The first component provides tool support for reporting defects within model reviews. This improves the review execution and reduces the effort for data collection. Additionally, the second component implements tool support for administrating reviews and experiments. The main goal of this second component is the provision of tasks to the experiment participants and the monitoring of their progress during the experiment to reduce the administration effort. For the evaluation of our results, we conduct a controlled experiment with about 80 participants in which we compare the advanced MDRE tool support with the traditional pen-and-paper-based approach and analyze the results.

The evaluation shows that the enhancements implemented into the MDRE provide similar results compared to the traditional pen-and-paper-based approach in all measured categories for both model reviews and the administration of experiments and reviews. Therefore, the improved state of the MDRE is capable of performing reviews similar to the traditional pen-and-paper inspection process. Furthermore, the implemented tool support for the administration of reviews and experiments reduces the effort and simplifies the process. Therefore, the MDRE helps reviewers, review managers, and experiment managers to plan, execute and manage reviews and experiments with fewer resources required.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung	xi
Abstract	xiii
Contents	xv
1 Introduction	1
1.1 Context & Motivation	1
1.2 Problem Definition	3
1.3 Aims & Expected Results	5
1.4 Thesis Structure	6
2 Related Work	9
2.1 Software Development Life Cycle	9
2.2 Model-driven approaches	15
2.3 Software Quality Assurance	19
2.4 Inspections (Reviews)	21
2.5 Software Inspection Tools	24
3 Experimentation	33
3.1 Principles of Investigation	35
3.2 Attributes & Types of Experiments	38
3.3 Experiment design & execution	39
4 Research Questions & Solution Approach	45
4.1 Design Science	45
4.2 Prototype for tool-supported model reviewing (RQ1)	47
4.3 Review and Experimentation Process Support (RQ2)	48
4.4 Benefits and limitations of MDRE (RQ3)	49
5 Model Design & Review Editor	51
5.1 Initial State of the MDRE	52
5.2 Tool Support AddOn for Model Reviewing (RQ1)	64
5.3 Tool Support AddOn for Experimentation (RQ2)	73
	xv

6 Empirical Evaluation (RQ3)	91
6.1 Study Process	91
6.2 Results	105
7 Discussion & Limitation	113
7.1 Discussion	113
7.2 Limitations	115
8 Conclusion & Future Work	117
8.1 Conclusion	117
8.2 Future Work	119
List of Figures	121
List of Tables	123
Bibliography	125
Appendices	134
Appendix A: Flow chart MDRE review execution	134
Appendix B: Flow chart MDRE experiment administration	135

Introduction

In this first chapter, we initially present the context and the motivation for this work. We also explain the problem definition, aims, and expected results. At the end of this section, an overview of the individual chapters is given and the structure of this work is presented.

1.1 Context & Motivation

In modern software engineering, the process of software development starts by defining requirements, designing the artifact, and continues with the implementation and testing. Even after many decades of active research in this field, it does still not seem to be possible to deliver software that is completely free of mistakes. [47] Since software developers are also just human beings they tend to make mistakes when implementing software features, which are called bugs or defects and often remain hidden within the software until they are found and later corrected. [81]

To address the problem of defects in software artifacts, Software Quality Assurance (SQA) is used. [20] SQA is a structured approach that aims to increase software quality by preventing defects before they are implemented and tries to find and remove already implemented defects using multiple different techniques. [11] Defects in software artifacts can be detected either dynamically by running tests during runtime or statically by reviewing the design artifacts or source code without running the software artifact itself. [81] To find defects as early as possible models are used as artifacts in the design stage of the software development. [16] These models are a simplified view of the real software artifact that should be implemented and are therefore easy and cheap to develop. [21] With the help of model reviews, it is possible to find defects that would be discovered earliest during the implementation already within the design of the software artifact. Finding a defect during the design phase is 6.5 times cheaper than during the implementation and 100 times cheaper than finding it when the software artifact is already in productive use.

[16] Therefore, models are an important concept of the software engineering process and are also a big part of SQA.

However, reviewing models during the software development process is not a trivial task to perform. Many different review methods can be used to review a model. Some of these review methods are supported by tools and others are not. However, there is only limited tool support available for reviewing models and it can be of importance to compare different review strategies with each other. This comparison can be achieved with the help of a scientific study like an experiment in which two or more review methods are compared and scientifically verified results are generated. Also, for this comparison, there is only limited tool support available, and therefore comparing two review methods during an experiment can be a resource-intensive task.

An illustrating use case for this work consists of an experiment at the Technical University Vienna¹ to evaluate the differences of two or more different review strategies by reviewing one or more software artifacts for defects and comparing the results of each strategy that is used. The experiment addresses both the scientific software engineering community with the experiment administration and design, as well as the industry by inspecting software artifacts for possible defects. Within such an experiment there are multiple stakeholders involved:

- **Experiment Managers:** The experiment managers are part of the scientific software engineering community. They are responsible for planning the experiment and the executed reviews, inviting participants, and evaluating the results from the comparison of two review strategies. The main point of interest for them is the advantages and disadvantages of the compared review strategies to each other.
- **Review Owners:** The review owners are responsible for the planning of each review and the execution of the review strategy itself. They are part of the industry and mostly interested in the defects which are found during the reviews.
- **Reviewers:** They are the participants of the experiment and have to conduct the different review tasks described in the materials which are provided with the experiment. Every reviewer inspects one or more artifacts with one or more review strategies during the experiment.
- **Model Designers:** The models that are inspected by the reviewers within the software inspection are created by model designers. This role is responsible for correcting the defects that have been reported by the Reviewers during their review. After phase E5 in which the results of the experiment are packed and presented the review managers forward the reported defects to the model designers.

¹Technical University Vienna: www.tuwien.at (last visited 30.09.2021)

1.2 Problem Definition

The main problem of our use case which we can see in the high-level view task flow displayed in figure 1.1 is that many different steps are involved, which are at the moment distributed over many different platforms. As of now, data must be stored in many different spreadsheets using solutions like Microsoft Excel² or Google Sheet³. There currently is only limited tool support that provides support for reviews and experiment administration at the same time. Our main motivation is to solve these problems and to implement a software solution that provides full tool support for most parts of the experiment and review administration, as well as improves the review process. The tool support which is developed should be capable to form groups and assign different tasks to each group which have to be carried out by the participants of each experiment group. Along with the general support for experiments and the centralization of the whole review planning and experiment administration process into one application also existing and new review functionalities should be integrated into the experiment administration tool support. In the next paragraphs, we explain each challenge of this work individually. Furthermore, the figure 1.1 shows all stages of our experiment use case and assigns the stages to their challenges.

Challenge 1: Limited tool support for model review. The main problem domain is the comparison of review types when used on models. There are many different review types but there is only limited tool support available to ensure that the reviewers follow the review method. The comparison of two or more review types can for example be achieved by measuring different metrics like the average time until the first defect in a model has been found, the found defects, the time until participants finished the review, and so on. Again, there arises the problem of missing tool support for measuring the previously mentioned metrics, which makes a comparison of review techniques difficult.

We use the *Model Design and Review Editor* [77] (MDRE) that is described in detail within chapter 5. For example, the MDRE does not record any found defects as of now, but only stores for each review if there were defects found or not by a reviewer in a certain part of a model. But due to the MDRE being a universal platform that supports every model from every domain by using an image of this model, advanced support for reporting individual defects solves this shortcoming. It also opens the possibility of supporting all kinds of different review strategies and evaluating the results from them.

Challenge 2: Limited tool support for review planning and experiment administration. As already explained in our example use case, review type comparisons are usually done with the help of an experiment with groups of participants needed to perform reviews on provided base models. All groups review the same model using different review techniques, which allows collecting and analyzing data as the foundation

²Microsoft Excel: www.microsoft.com/de-de/microsoft-365/excel (last visited 30.09.2021)

³Google Sheets: www.google.com/intl/de_at/sheets/about (last visited 30.09.2021)

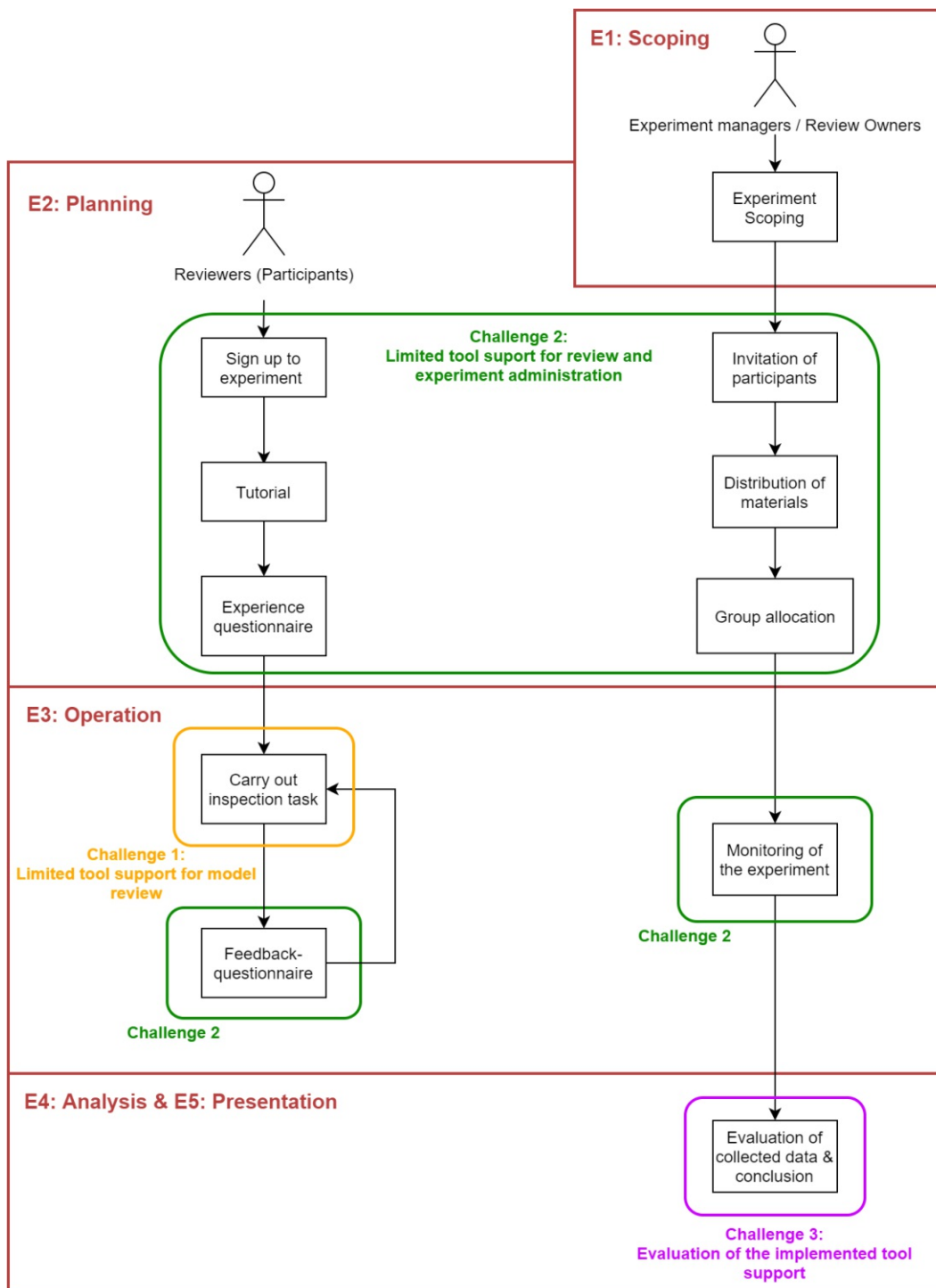


Figure 1.1: Reviewing and empirical study process challenges

for an overview of the pros and cons of each review type. Therefore, participants are usually assigned to a group with a certain review type. Without tool support, the task of splitting up the participants into individual groups with different review techniques can quickly become time-consuming. In addition, participants often have to answer questionnaires and fulfill other prerequisites before starting the experiment. This adds management effort to check if all prerequisites are fulfilled.

In the current state of the MDRE, the preparation of an experiment with 50 and more participants would quickly turn into a time-intensive task, as the MDRE currently does not support experiment execution and planning. But the MDRE as a whole provides a universal platform that could be used to achieve tool support for experiments. Advancing the platform with experiment tool support should make experiments in which multiple review methods are compared less time-consuming and easier to manage, reducing the overall administrative overhead.

Challenge 3: Evaluation of the implemented tool support prototypes It is important to show that our research provides benefits and limitations. For example, practitioners from the industry need a way to find out which improvements and limitations our new approach has before they can implement it into their industrial processes. But also scientific researchers need the evaluation of our research results if they want to use or enhance them within their research. Therefore, the developed tool support components for the MDRE approach need to be evaluated.

This evaluation needs to be achieved by conducting a study in which the advanced MDRE approach is compared to an existing review method without tool support like the traditional pen-and-paper-based inspection process. The research results that are collected during the study need then be evaluated and put into comparison.

1.3 Aims & Expected Results

We focus on advancing the currently limited tool support for model reviews, as well as for review and experimentation administration. Therefore, we use the MDRE in its initial state and aim to extend it with two tool support prototypes. The first component should provide tool support for reporting defects within model reviews and simplifies the data collection during the execution of these reviews. Additionally, the second component implements tool support for administrating reviews and experiments. The main goal of this component is to provide tasks to the experiment participants and monitor their progress during the experiment automatically.

To verify the implemented components it is necessary to collect data and perform an evaluation. It should be possible to execute an experiment with the MDRE in which two different review methods are compared to each other. With the help of this component, there should be less effort required in the setup, execution, and monitoring of the experiment. Therefore, we conduct an experiment, with the newly implemented experiment administration component, to evaluate the advanced MDRE approach. Finally,

the results of the executed experiment are evaluated, put into comparison, and a conclusion is drawn. As the main result, we expect that the further developed MDRE approach is performing at least as well as the pen-and-paper-based approach or better in the majority of all result categories. In figure 1.2, a summarized overview of the different work packages and dependencies is displayed.

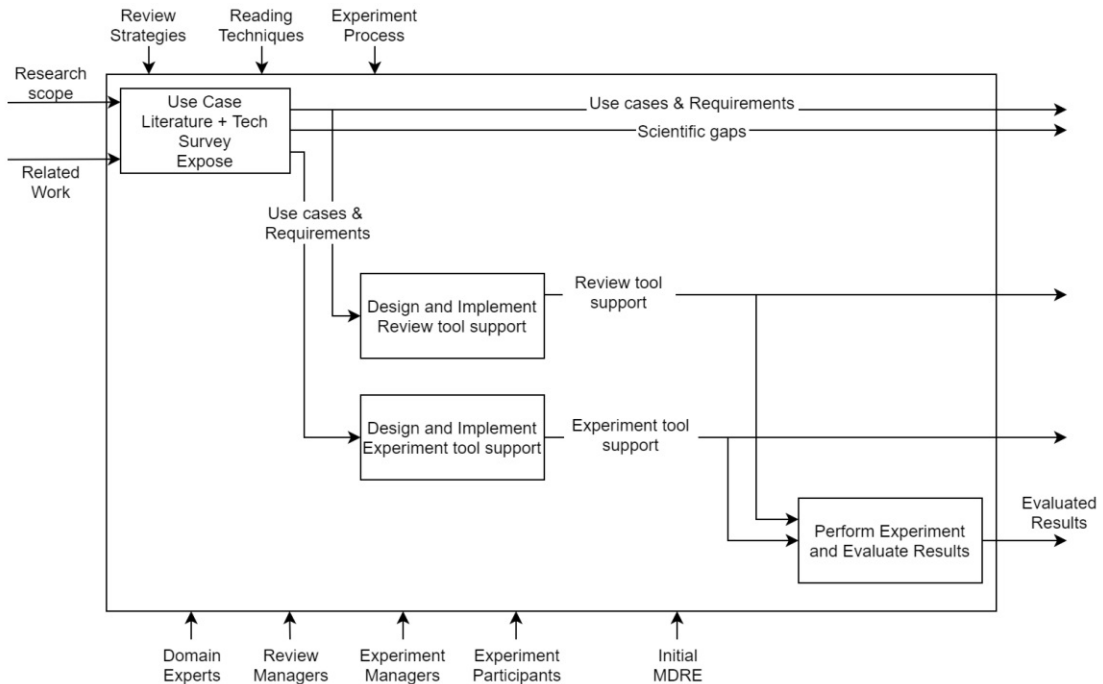


Figure 1.2: Overview of the work packages of the solution approach

1.4 Thesis Structure

Chapter 1 presents an overview and explains the problems, aims, and expected results, as well as the motivation and the context. The main goal of this chapter is to provide a general understanding of the topics and goals.

In chapter 2, the state of the art and the most important concepts about the topic are explained. Furthermore, the related work is used to build upon this topic and to develop the solutions for the research questions within the next chapters. In this chapter, a basic overview of models, software quality assurance, and model-driven development approaches are given. Additionally, various other software inspection tools are presented and their advantages and disadvantages are explained.

Chapter 3 is another state-of-the-art explanation that especially focuses on experimentation. Within this chapter, the most important details about scientific studies and experiments are presented. The explanation of experimentation is important for the

upcoming chapter 6 in which an experiment is executed to evaluate the results of this work.

Within chapter 4, the research methodology called Design Science, which is our main methodology is explained. Furthermore, the three research questions for the development of two tool support prototypes and the evaluation are defined. Within this definition of the individual research questions, the solution approaches on how the research questions are solved in detail are presented.

In chapter 5, the MDRE that is the main focus of this work is presented. Since the MDRE has already been in development previously to this master thesis, we first explain the initial state of the MDRE and why it has been developed. Furthermore, the two tool support components which are developed to advance the MDRE approach are presented afterward. The main focus of the presentation of these new components is explaining the reason why it has been implemented by comparing the MDRE to existing software inspection tools and presenting the functionality of each individual component.

Chapter 6 contains the evaluation of all research results. First, the study design of the controlled experiment that is conducted to evaluate the developed tool support prototypes is explained. After the study design is explained, the results that could be collected during the execution of the controlled experiment are evaluated and put into comparison. During the evaluation of every individual result category, an explanation is given that explains which of the inspected methods performed better within the category.

In chapter 7, the results are discussed and possible limitations of the newly implemented tool support prototypes are presented.

Within the final chapter 8, a conclusion about the results and all topics is drawn. Furthermore, possible future work that could not be carried out but could be interesting in the future is presented.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Related Work

In this chapter, the state of the art around this topic is explained and related works are presented. The main goal of this chapter is the definition of the research gap that we want to address.

2.1 Software Development Life Cycle

The process of Software Engineering advanced over the last 30 years, but even with all the achieved improvements large and complex software projects still experience several changes during their development. Most of these changes cannot be prevented because of constant changes in the ecosystem around the software. [48]

The changes are unpredictable and derived from several factors like constant changes in utilized software, system requirements, business goals, market demand, work environment, and government regulations. All these mentioned changes in the requirements can produce significant project uncertainty and have been reported as one of the main factors why projects become challenging. [48]

Software is designed, developed, and maintained in multiple different stages which are part of the life-cycle of the software product. [39] This life-cycle is therefore called the Software Development Life Cycle (SDLC). The SDLC covers all different stages in the software development process starting with gathering requirements and designing the product over development and testing to the productive operation of the product. [16]

The general SDLC which contains every step within the software development process can be shown as a simplified and theoretical model as it is presented by Shylesh [63] and by Mishra et. al. in [45].

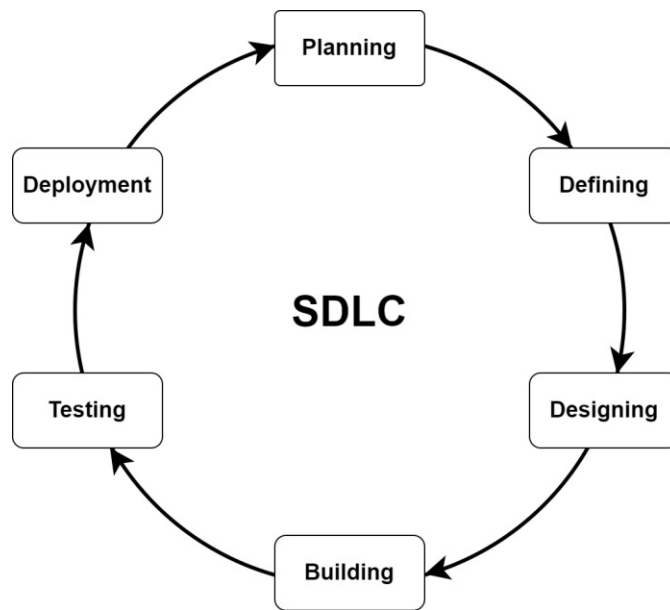


Figure 2.1: Steps in the SDLC [63]

The provided model, which is displayed in figure 2.1, defines the SDLC with only 6 different phases which are:

- **Planning:** During the planning stage requirements are analyzed along with the customer or owner of the software system. The main outcomes of this stage are quality assurance requirements, risk identification, and a feasibility report. [63]
- **Defining Requirements:** In this phase the requirements are defined in detail and documented to get a verification from the customer. The main outcome of this stage is the software requirement specification (SRS). [63]
- **Design:** In the design phase multiple software architectures are designed using the previously created SRS as input. The best design based on risk assessment, robustness, design modularity, time, and cost efficiency is selected. [63]
- **Implementation:** During the implementation phase the development of the software product is performed according to the previously specified design. [63] During this stage, the design documents are converted into code with the help of one or more programming languages. The output of this stage is a working software application in the form of program code. [37]
- **Testing:** In the testing phase the previously developed software product is tested against the in the first stages created specification documents like the SRS. All discovered software defects are reported, tracked, and fixed within this stage. [63]

Effective testing results in high software quality, lower maintenance costs, and reliable results, therefore this is the most important stage of the SDLC. [37]

- **Deployment and Maintenance:** After the testing phase has been completed successfully the product is deployed in the destination environment and the customer performs an acceptance test. [63] [37]

It is important to point out that this model is a simplification of the SDLC to give a brief overview and so it is only a theoretical concept. It is not a model which would be used as a process model in a real software development project as it is not advanced enough. [63]. [16]

Every SDLC model consists out of several finite steps which are used during software development. There is not just one single model implementing the SDLC in higher detail, but there is a bigger amount of models.

Some well-known examples of models implementing the SDLC are the waterfall model, iterative model, V-Model, rapid prototyping model, spiral model, scrum, extreme programming (XP). [55] [15] The most important models within this list are described in detail in the next sections.

Every individual SDLC model defines a process on how the different stakeholders shall proceed in the different phases of a software development project. There also is not one perfect model having an advantage over all other models, every model has its own method with all the pros as well as cons. In some models, the modeled SDLC consists out of 10 and more sequentially connected phases and some other models are even partially iterative, which leads to a bigger number of different steps in the life-cycle. [58]

Currently, there are not just different SDLC models but there are also two different SDLC methodologies to which the models belong. These two methodologies are called traditional software development and agile software development. [41] In the next section, we explain both methodologies and point out the differences between the individual methodologies.

2.1.1 Traditional Software Development

In traditional software development, the SDLC models are based on a sequential series of steps like requirements definition, solution building, testing, and deployment. An important key aspect of this methodology is that it requires the definition and documentation of a stable set of requirements at the start of a project, in which the requirements do not change at all during the whole project. [41]

Therefore, the success of the whole project which is executed with the help of the traditional software development methodology relies on the knowledge of all the requirements of the whole project before the beginning of the development in the implementation stage. [41] A change during or after the implementation could be problematic and lead to additional costs. On the other hand, if the requirements do not change during

the project the traditional software development offers the possibility to determine the costs, as well as the time and the required resources of the project easier than in other methodologies. [41] In the next section, we describe two important models for traditional software development namely the waterfall model and the spiral model.

Waterfall Model

The waterfall model was originally defined in 1970 and later improved in 1976 to cope with the growing complexity of software projects. [15] Since that time the waterfall model has not been altered much and is still mostly similar today. [6]

The waterfall model consists out of different consecutive phases that can only be completed one after another. Moving to the next stage is only possible if the preceding stage is completely done. This makes the waterfall model recursive because in this model every stage can be repeated an unlimited amount of times until the produced software artifact is perfect. [6] In figure 2.2, the different phases of the waterfall model are displayed.

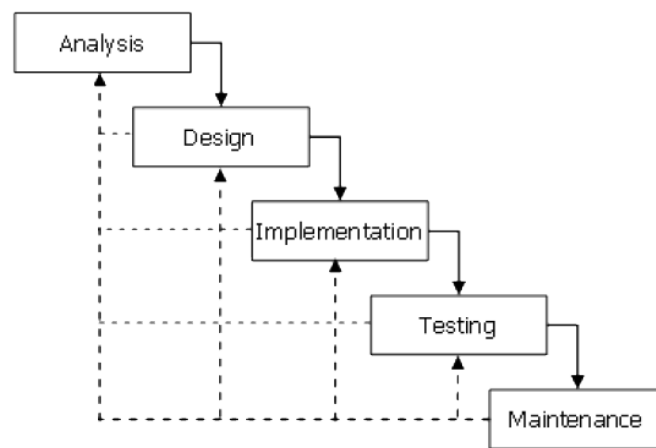


Figure 2.2: Phases of the Waterfall Model [6]

The **advantages** of the waterfall model are that it is easy to understand and it uses good habits like define-before-design and design-before-code. It also includes the creation of different deliverables and the definition of milestones to highlight the major goals in the project. [55]

On the other hand, the **disadvantages** are that the idealization does not match the reality as the constantly changing environment around every software development project is not reflected. [55] Software is delivered late in the project which delays the discovery of critical defects in the developed software. In small teams, the waterfall model results in overhead and increased costs for project management. [55]

Spiral Model

The spiral model is a modified version of the waterfall model and was introduced in 1986 by NASA. [58] This model introduces several iterations that spiral out from the center of the model, starting small and becoming larger as the progress continues, which represents the recommended project management strategy of starting small and going bigger as time passes. Figure 2.3 displays the process and the different stages within the spiral model. In the spiral model, the focus is shifted from a specification-driven approach (waterfall model) to a risk-driven one. [58]

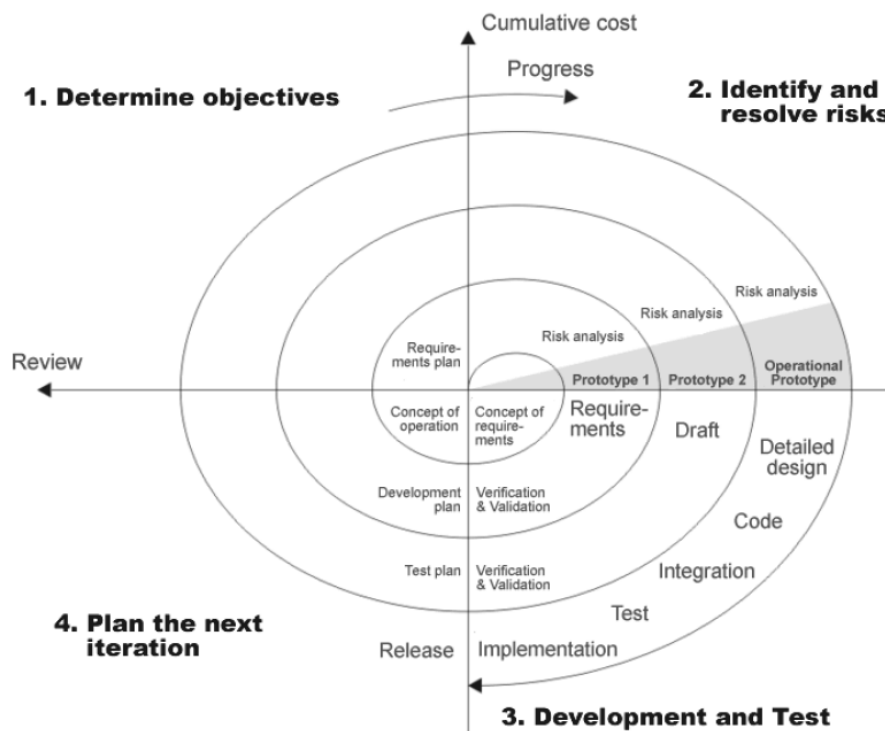


Figure 2.3: Phases of the Spiral Model [58]

With each spiral progression, a prototype is built, verified against its requirements, and validated through testing. This makes it possible to find defects in the developed software earlier than in the waterfall model and offers the possibility to fix them in the next progression of the spiral. This reduces the additional costs which arise because of the defect. [58]

The **advantages** of the spiral model are the high amount of risk analysis which makes it especially good for large and mission-critical projects. The fact that software is produced early in the SDLC can reduce costs in the case that requirements change or defects are discovered. [55]

The **disadvantages** are that it can increase the costs for the project management in

comparison to the waterfall model. Risk analysis during the project planning requires specific expertise and the project success highly depends on this risk analysis. This also means that the spiral model does not work well in smaller projects. [55]

2.1.2 Agile Software Development

Agile software development combines the ideas of incremental and iterative development. Instead of formalizing the requirements of the software product in a detailed specification [60], this methodology revisits the phases within the SDLC over and over again which improves the software product in every iteration of the cycle. [41] Additionally, with every development cycle, a working software application is developed and customer feedback is collected, which is used to improve the software in the next cycle. [41] The production of software can be complicated, therefore the agile approach focuses on communication, flexibility, and good analysis. [37] The Agile Manifesto defines key principles for agile software development which are early customer involvement, iterative development, self-organizing teams, and adaptation to change. [41]

Scrum

Scrum is a lightweight project management framework that key aspects were introduced in 1986. [60] [37] It defines a scrum team as a group of around ten people who have different roles. [55] In every scrum team, there exists a scrum master and a project owner. The scrum master is responsible for the whole scrum team, although the scrum master has no supervisor role. [60]

The scrum master role is responsible that the scrum process is adhered to, for handling organizational aspects, and that problems that might hinder the team from developing are eliminated. [60] The product owners are not the customers themselves but part of the scrum team. They define the team's development targets for the next sprint and are responsible that valuable software is produced for the customer. All other scrum team members are developers who implement the features which are on the agenda for the running sprint. [60]

The sprints split the whole development process into small cycles which are usually 1-4 weeks long. After every sprint, the team has to deliver working software according to the agile manifesto, which the customer can review. The team handles a product backlog that contains all features for the whole software development process in a prioritized order. [60] For every sprint, an actionable amount of features is moved from the product backlog to the sprint backlog. All features which were moved to the sprint backlog are then fully implemented within the upcoming sprint. [60] Figure 2.4 displays the whole scrum development cycle in detail.

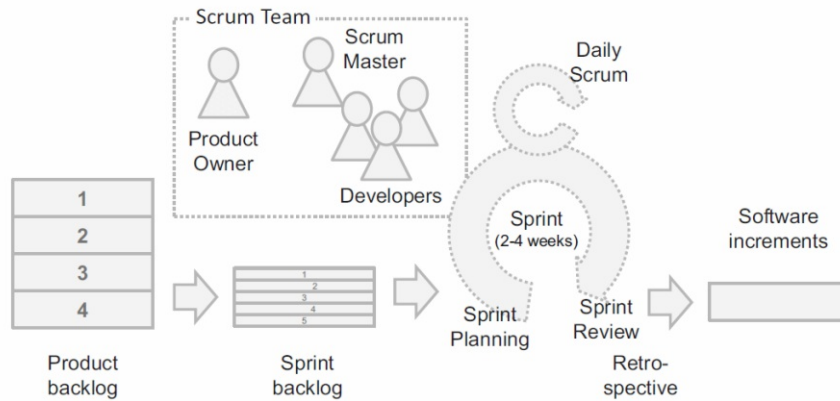


Figure 2.4: Phases of the Scrum Model [60]

2.2 Model-driven approaches

Software engineering faces the challenge that it is still quite new and immature compared to other engineering disciplines. Due to the current pressure of the software market users often demand complex applications of which the construction of such software systems exceeds our abilities. [61] At the same time, the complexity of software is growing exponentially, leading to the general agreement that managing the growing complexity can only be achieved by using appropriate methods of abstraction. [25] Furthermore, the whole software engineering field seems to be evolving slower than any other engineering discipline. Since the invention of the first programming languages in the 1950s programming languages haven't changed as much as expected. Even with modern programming languages like Java or C++ the basic concepts like loops and if-statements are still the same as they were more than 50 years ago. [61]

Since nowadays a big revolution in software programming languages is not expected any longer, there is a shift from focusing on technical improvements to process improvements. [61] This is also the main reason why the previously explained software development life cycle models like scrum are popular nowadays. Software development, in general, consists a lot out of expressing ideas, which is rather limited by the imagination of the development team than by the underlying physical laws. To make sure that the software system is not becoming too complex for our imagination they can be described by a model-driven approach instead of just using program code, which also improves the software development process, while the technical programming languages stay untouched. [61]

2.2.1 Models

Models are a simplified view of the reality [21] and so they are a key concept in many scientific contexts, e.g., when developing modern software or in physics, chemistry, and

mathematics. [9] Furthermore, they provide an abstract view of a system and so offer the possibility to ignore extraneous details and give engineers the ability to focus on the relevant parts of the system. [10] Many aspects are relevant when a system is modeled, for example, different modeling concepts, perspectives, views, and notations can be used at any point in the time of the development by various stakeholders. In addition, the model transformation of the different types of models is an important concept of models. [10] For example, it can often be needed to convert different views on the same level of abstraction, which is supported by certain model transformations. [10] In addition to the definition of a model it must conform to the following five important key characteristics: [21]

- **Abstraction:** Models must always be a reduced rendering of the system that is represented. [21]
- **Understandability:** It is not enough to remove detail from the system, the remaining parts must be in a form that can be understood. [21]
- **Accuracy:** The model must represent an accurate view of the modeled system's features of interest. [21]
- **Predictiveness:** The model must have the correct properties to predict the behavior of the represented system accurately. [62]
- **Inexpensiveness:** A model must be a lot cheaper and easier to create than the system which is represented by the model. [21]

In general, the whole world is full of models and the term "Everything is a model" is the common worldview for model engineers. [21]

Therefore, models enable at least two different roles of abstraction: [9]

- **Reduction feature:** The model has a clear focus and only contains the relevant parts.
- **Mapping feature:** The model represents an original product in an abstract and generalized representation.

Furthermore, there are also various purposes for which a model can be used for: [9]

- **Descriptive purpose:** Describes a real system or context which already exists.
- **Prescriptive purpose:** Determining the scope and details about a problem to study.
- **Definition purpose:** Defining how a planned system shall be implemented.

2.2.2 Model-Driven Architecture

At the end of 2000, the Object Management Group (OMG) invented a new standard approach called the Model-Driven Architecture (MDA). [7] The MDA paradigm invents several models which are used to represent a software system in multiple abstraction levels.

In the MDA approach, the first model that should be created in the software development process is the computation independent model (CIM). [50] The CIM focuses on the viewpoint of the system and considers the environment as well as the system requirements, while it does not display any information about the computerization of the system. [19]

In the next abstraction level, the so-called platform-independent model (PIM) is created. This model displays the parts of the system which are computerized but it does not define the technological platforms that are used to deploy the software implementation. [19]

The last model in the MDA approach is the platform-specific model (PSM) which is needed to receive a platform-specific view. In the PSM the system relying upon the specific characteristics of the platform that it is deployed on is described. [19]

Finally, the code model is created based on the PSM. Creating the code from the start point which is represented by the CIM model can be strenuous because there is a large number of models which need to be created on the way to the final code. [50]

2.2.3 Model-Driven Engineering

An important principle used in many engineering disciplines is called Model-Driven Engineering (MDE). [21] The use of models in traditional engineering disciplines like civil engineering, for example, uses mathematical models to calculate the forces acting within their structural design. [62] MDE also includes different model-driven approaches in the engineering disciplines like MDA, domain-specific modeling and model integrated computing. [25] MDE uses models as first-class entities and interprets every artifact as a model or as a smaller part of a model (model element). [7]

The MDE approach suggests that during the design phase the developers should first create a model of the artifact which shall be created before the actual development process has started. During the creation phase, the resulting model from the design phase is then be turned into a real-life product. [21]

In MDE not just the existence of models is mandatory but also the quality of the models which are used in the creation process is of high importance. Models in the MDE approach have two important quality criteria: [46]

- **Transformability:** Every model must have the ability to be transformed into another model type, which provides a higher amount of detail to executable types of code. [46]

- **Modifiability:** In case there are changes to the artifact on which a model is based on, the new requirements need to be rendered correctly in the model and also be adapted to the code. [46]

2.2.4 Model-Driven Development

The Model-Driven Development (MDD) approach is a subset of MDE. MDD focuses on the development part of the software engineering process, by providing support for generating implementations from models. MDE in comparison supports other uses of models within the software engineering process, for example, model-driven reverse engineering and model-driven evolution. [73]

The idea of models, modeling, and model transformation plays an important role in the MDD approach. In MDD models are used to argue about the problem and solution domain in software development. [10] However, models of software are often inaccurate because they are usually not open for abstractions. [62] Additionally, with the conversion of models converted into code by hand, there is no formal link between the produced software product and its models. Therefore, the model and the developed software often drift apart, as the overhead which is required to keep the two artifacts in line is seemingly too high for most development teams, as there is no visible value from this work in the first place. [62] As an outcome, the models are usually incomplete and are not trustworthy documentation of the software system. Another problem is that many software languages, especially older ones, have weak semantic specifications, which leads to unclear and different modeling styles. [62]

Nowadays, the main focus of the MDD approach is therefore the automatic code generation from models, which are formally linked to the code without any further developer interactions needed. [62] Over the past decades, technologies have evolved to fulfill the potential of MDD. With the new improvements in MDD technology, a viable alternative to traditional development processes has finally been found for most application domains. [62]

The main motivation for MDD is the improvement of productivity. Therefore, there are two important benefits which are delivered in the following way: [2]

- The short-term productivity of the software developers is improved with the help of an increase in the software artifacts value in terms of how much functionality it delivers. [2]
- The long-term productivity of the software developers is improved along with the reduction of the rate at which the software artifact becomes obsolete. [2]

Model-driven development requires tool support that combines the creation and design of models, as well as, the review of existing models to make sure that there are no hidden defects. However, there is currently only limited tool support available that combines

the possibilities of designing and reviewing models within the same tool. Often, different tools are used and a lot of effort is required to export and import individual models between the used tools.

2.3 Software Quality Assurance

In this section, we present relevant information on software quality, software quality assurance, and inspections (reviews).

2.3.1 Software Quality

Software quality is quite difficult to define and capture to its whole extent. [72] People often see software as a simple set of instructions that form the lines of source code and make up a software application. But a software application is only a small part of a whole software system, which the application interacts with nonstop. [40] Therefore, it is not sufficient to see software quality just as the quality of the source code of the software application which is executed within the system. We need to identify all parts of the software system and their ways of interacting to ensure high software quality. [40]

Due to the wide and often blurred scope of quality, there is not one correct definition of (software) quality. Gaffney et. al. in [20] describe quality as conformance to requirements. In this definition, conformance means that the product meets the needs of the users and satisfies the performance criteria, which has been agreed upon with. The needs and requirements of the user to the product should be provided in a written form to the developers before the design phase in the SDLC is started. [20]

Software quality has always been a central role in the SDLC and becomes more and more important every day because with the evolving technologies people are increasingly affected by software systems in their everyday life. [72] Nowadays we are constantly online on social networks like Facebook, Twitter, and Instagram. Important systems like acceleration, steering, braking, safety systems like the airbag, navigation, etc. are controlled by software. [72] Mistakes in software lead to unexpected/unplanned behavior of important systems in our life, which often ended in severe consequences in the past, like high costs, leaks of private information, and people who died. In summary, defects in software can be traced back to bad software quality and problems within software quality assurance. [72]

In software quality, the term defect has a general meaning and refers to some kind of problem with the software either with its external behavior or its internal characteristics. The term defect is split up into three different categories by the IEEE Standard 610.12 (IEEE, 1990) [69]. The meanings of these categories are related but there are important distinctions, which we present in the following section: [47] [69]

- **Failure:** When the external behavior of a system does not match the expectations which have been prescribed within the specifications of the system we speak of a failure. [47]
- **Error:** An error can lead to the failure of a system if there are no corrective actions applied. An error is considered a possible state of the system. [47]
- **Fault:** The main cause behind an error is called a fault. [47]

In summary, this means that a fault is hidden in the code and can stay undetected for a long time until it is activated by an event [47] or found during a software quality assurance measure. In case that the fault is activated the whole software system enters an error state and if there is no corrective action applied the software system eventually causes a failure. The general defect chain is fault \rightarrow error \rightarrow failure, but a fault does not automatically lead to an error and also an error does not lead to a failure each time. [47]

2.3.2 Software Quality Assurance

Since defects in software systems can have serve consequences it is necessary to find and prevent them. Therefore, the field of software quality assurance (SQA) has been invented [11], with its main focus on keeping the software quality up. [20] By definition, the term quality assurance is defined as: *"A planned and systematic pattern of all actions necessary to provide adequate confidence that the item or project conforms to established technical requirements"*. [11] However, the term SQA can be misleading because it cannot prevent defects in every case for sure. No matter how seriously executed and advanced SQA measures become there never can be an "assurance" to prevent defects in software systems completely. SQA can only "assure" the quality of a software system but it cannot give confidence. [40] To increase the software quality, SQA uses multiple activities to deal with certain defects, which can be grouped into three different categories: [69]

- **Defect prevention:** The aim of this SQA activity is the prevention and avoidance of the injection of certain defects into the source code of the software system. The defects which are prevented in this activity are usually caused by the humans developing the system. Some examples of defect prevention are training for developers, the introduction of formal methods, the introduction of standards and processes, and the usage of certain tools and technologies to increase code quality. [69]
- **Defect detection & reduction:** This SQA activity detects and removes certain defects which have been implemented into the software system. Defect reduction can be achieved using processes for software verification. [13] This category contains the two best-known SQA activities which are inspections (reviews) and software tests. [69]
- **Defect containment:** In this SQA category defects are expected to appear and so this category aims to either contain the defects to a local area so that there are

no global consequences or to invent mechanisms that at least reduce the damage caused by a failure in the software system. This is achieved by fault-tolerance techniques which break the relation between faults and failures so that a fault does not cause a global failure. Another possibility is to introduce containment measures that prevent serious consequences like death and injuries in the case of a software system failure. [69]

The prevention, reduction, and containment of defects, which would normally stay undetected and be triggered later, can save a lot of costs and resources if SQA is used at the right time within the software development process. [16] Therefore, it is important to find defects as early as possible during the SDLC. A defect, which is not detected in the model during the design phase, can lead to high costs, which arise from fixing the defect during or after the implementation phase. According to Dawson et al. in [16] a defect found during the implementation is 6.5 times more costly than one which is found during the design and a defect found during the operation is 100 times more costly. One example of an area in which SQA has an important role in model-driven engineering. In this field models must be sufficiently complete and correct, i.e., the model does not have serious issues, which would jeopardize the successful production of the artifact derived from the model or require significant additional effort.

We focus on the SQA category of defect reduction. The aim of the defect reduction activities is the verification of the artifact of which the quality should be increased. Verification is an activity that is used to confirm that an artifact conforms to the specified state which has been documented in the specifications. [13] In general, it is the confirmation that the right product has been built. Different tools can be used to verify an artifact like walkthroughs, inspections (reviews), and audits. [13]

2.4 Inspections (Reviews)

Inspections are part of our main focus. The procedure of an inspection is usually described with the help of certain materials and documents (e.g. checklists and printed forms) which are provided to the reviewers who perform the inspection tasks. [49] In these provided materials the exact tasks are described, with the main goal that the inspection is so careful and complete that if an inspection cannot reveal any defects the confidence that the product performs as expected is justified. [49] Inspections can not just be applied to check product artifacts like software applications and their source code but also to check various design documents like models and specifications. We focus on inspections that eliminate defects early in the SDLC which leads to a reduction of the required resources. Therefore, inspections are an important activity in early software development. [76]

According to Laitenberger in [38] the nature of software inspection can be characterized in four different primary dimensions which are called *technical*, *economic*, *organizational*, and *tools*. Each primary dimension has a major goal and several different sub-dimensions. [38] Figure 2.5 displays all possible dimensions and sub-dimensions of software inspections.

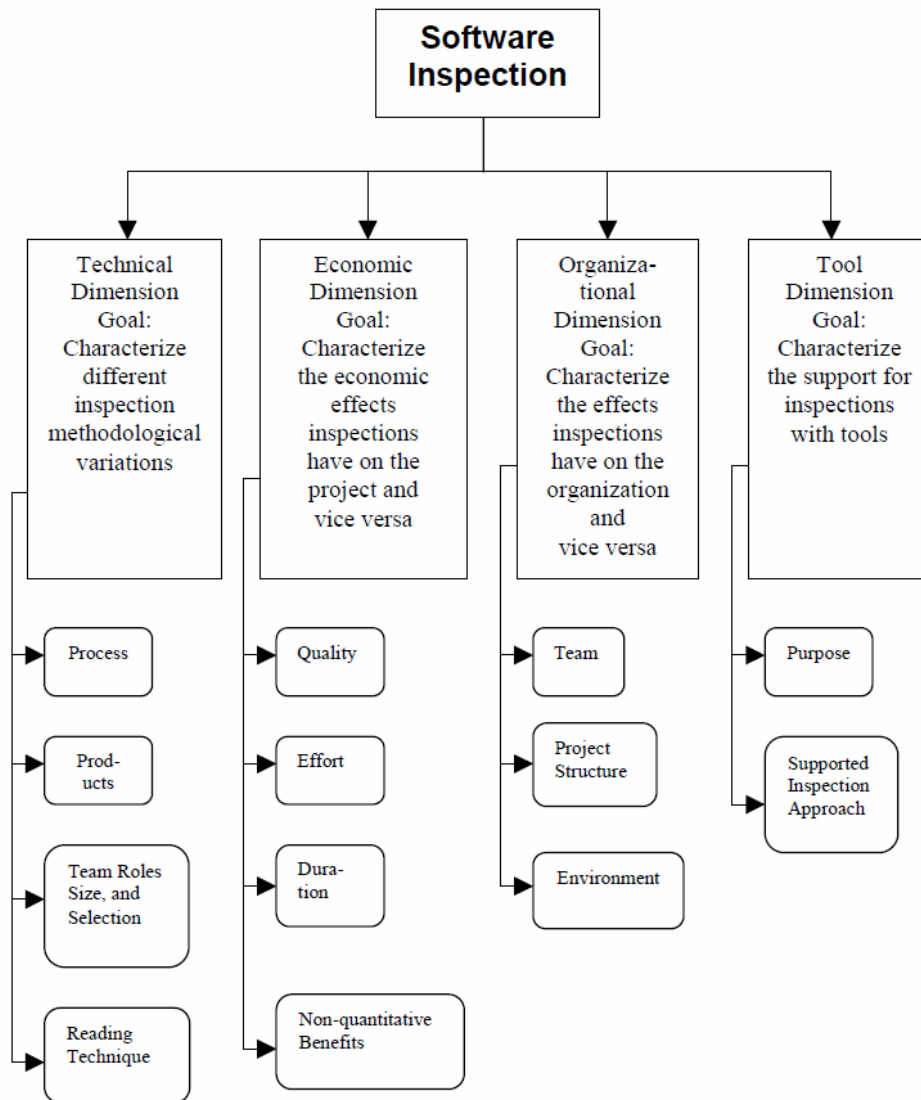


Figure 2.5: Dimensions and sub-dimensions of software inspections [38]

- The **technical dimension** characterizes the differences between inspection methods to identify similarities and differences which are among them. Each inspection approach can be characterized in different technical sub-dimensions. [38] These subdimensions are the activities performed (process), the software product which is inspected (product), team roles, their optimal size and the participant selection (team roles, size, selection), and the technique used to detect defects in the product (reading technique). [38]
- The **economic dimension** is needed to receive information about the project's effect on the inspection and vice versa. Most important for these dimensions are the

sub-dimensions which provide information about the product quality (quality), the project effort (effort), the project duration (duration), and the contribution to the education of participants, as well as, the team building in general (non-quantitative benefits). [38]

- The **organizational dimension** collects information about the impact of the whole organization on the inspection and vice versa. Important sub-dimensions in the organizational dimension are the participants of the inspection (team), the structure of the project (project structure), and the environment around the project (environment). [38]
- In the last dimension, the **tools** used to support the software inspection are described. The sub-dimensions in the tool dimension collect information about the purpose of the used tools (purpose) and investigate how the used tools support the given software inspection approach (supported inspection approach). [38]

Due to the different requirements in SQA, not every inspection is performed after the same procedure. There are different inspection techniques available that can be used for checking an artifact on its correctness, for example, checklist-based reading and usage-based reading [68]. Since there are many different inspection techniques it can become necessary to compare individual methods to each other. These comparisons are performed within scientific evaluations like experiments. However, a problem with comparing inspection techniques within experiments is that the administration and execution of experiments is laborious. A possible improvement to this situation is the use of tool support that reduces the effort that is needed for executing the comparisons. Within section 2.5 we present various tool support components for administrating and executing inspections. In the following sections, we explain the previously mentioned inspection techniques in detail.

2.4.1 Checklist-Based Reading

During a review that uses checklist-based reading as a review method, a checklist with questions or imperative sentences which can be used as hints and recommendations for finding defects is used. [22] The questions and sentences should focus the reviewers' attention on the main key aspects of the reviewed artifact which is usually the area of the artifact which is most defect prone according to previously collected evaluations. [68] Checklists also make sure that the reviewer does not forget any aspects which need to be inspected. Findings and collected data from different or the same projects are used to develop the checklists for the reviews. But, the previously mentioned checklist type does only give little guidance on how the reviewers should proceed with the review. [22] [81]

Therefore, an advanced and modern type of checklists can be used for checklist-based reading, which are checklists with guidance. According to Winkler et al. in [76] checklists that give active guidance on how a reviewer shall proceed and actively lead the reviewer through the review are a good enhancement to checklists with only little guidance. In

this new form of checklist-based reading, every reviewer can follow an exact procedure with the help of the available checklist. This makes sure that there is no area of the artifact that is up for review is left out by the reviewer and that they are looking at the expected properties in the expected and prioritized order which was specified by the review managers previously during the development of the checklist. [76] [81]

2.4.2 Usage-Based Reading

The previously mentioned reading techniques focus on finding as many defects as possible in any area of the inspected artifact, without taking the importance of each area into account. In general, the efficiency of reviews is measured by the number of defects that have been found and usually ignores the severity of the final impacts of the found defects. [68] To find the most critical defects, that matter for users of the inspected artifact, usage-based reading was introduced. The main idea of usage-based reading is to define and prioritize use cases and focus the inspection towards identifying the critical defects within the important use cases of the artifact by providing active guidance.

A use case is a specification of an artifact that has several interaction steps that need to be executed and specific goals which should be achieved after executing all steps defined in the individual use case. During the review, the reviewers only inspect one use case after another and ignore all other functionalities of the artifact which are not part of the selected use case. [81] Due to the focus on the use cases, the understanding of the reviewers is improved by usage-based reading and therefore reviewers are supported in finding more important defects in the inspected artifact. [76] Two different schemes can be employed in usage-based reading:

- **Time-boxed:** In this reading scheme a fixed amount of time for the whole reading is chosen and the allocated time for every individual use case is divided according to the priority of the use case and the severity of a defect in this use case would have. [81]
- **Rank-based:** With this reading scheme the use cases are reviewed ordered by their priority and there is no time limit per individual use case but there is a total time limit for inspecting the combination of all use cases which are up for inspection. [81]

2.5 Software Inspection Tools

The most common method used in software inspection to assure the quality of a developed software artifact is code reviews. Therefore, code reviews are common in software engineering and used by organizations and (open-source) communities as a method to improve the quality of their software source code. [70] A code review aims to check if the code compiles based on the internal guidelines, to find potentials defects, and to improve faulty components. [70] These code reviewing activities result in the reduction of

broken builds, improves code quality overall, have a positive effect on the team's dynamic knowledge sharing, and allows an easier training of new members. [44] Reviews as a method in software inspections are not limited to reviewing code. [70] Many different types of artifacts can be reviewed by a group of peers, for example, documents, images, models, tables, files, and so on.

Nowadays, modern reviews are supported by different tools, which can manage the whole process of reviewing. [70] One of the first frameworks for review tool support is called ISPIS and was proposed in 2004 by Kalinowski et. al. in [33]. This framework provides the basic process steps for conducting a review with the help of a computational tool. [33] The authors later tried to advance ISPIS to an industry readiness in 2007 [34], but since then the development has not been continued and ISPIS has not been used in any later works after 2007. Today, many modern tools provide the support to review certain types of artifacts and many of them are not even focused on conducting reviews but are normal tools of daily use. Simple and well-known examples for software providing review functionalities are certain office and file-sharing tools. For example, in Microsoft Word and Google Docs, it is possible to enable a change history to mark all changes within a document and comment on certain parts within it. File sharing tools like Dropbox, Google Drive on the other hand provide the possibility to keep a change history and discuss the current stage via comments at the file level.

Both office and file-sharing tools usually provide review functionalities but it is not the main focus of these tools. Therefore, the review support within the tools is sufficient for single users or small groups but organizations need advanced reviewing tools. In the following section, we present a set of tools that focus on the support of reviews and which are used by groups and organizations of all sizes.

2.5.1 Review Board

Review Board¹ is an application written in Python using the Django web framework. [8] It is Open Source and published under the MIT license [8] and actively developed as the latest version 4.0.3 was released on the 29th of June 2021.

The general workflow while using Review Board is that in case a code review is done before a commit the review process starts when an author has made changes to the local source tree. [8] In the first step, the author starts by creating a diff-file comparing his changes with the codebase and submits this file to Review Board by creating a review request. After that, the author now has to wait until enough reviewers have provided their feedback for the change. [8] If the reviewers are satisfied with the state of the code then the author can submit the code to the trunk and the review is closed. In case there are changes requested by the reviewers the author needs to adapt to the changes and submit another diff-file afterward. This process is then repeated until all reviewers accept the change in the codebase. [8]

¹Review Board: www.reviewboard.org (last visited 30.09.2021)

2. RELATED WORK

In figure 2.6, the frontend of Review Board is displayed and in this case, shows the side-by-side code view.

```
django_evolution/mutations.py
Revision e5edcefb5fca7b8348f0cf08aff79cfe621a57b
New Change
+ 1137 lines
+20
def mutate(self, mutator, model):
1138     new_field_sig.field_attrs.pop('db_column', None)
1139
1140     # Create the mock field instances.
1141     Moved from line 1146
1142     new_model = MockModel(project_sig=mutator.project_sig,
1143                          app_name=mutator.app_label,
1144                          db_name=mutator.database)
1145
1146     new_field = create_field(project_sig=mutator.project_sig,
1147                             field_name=self.new_field_name,
1148                             related_model=new_field_sig.related_model,
1149                             parent_model=None)
1150
1151     new_model = MockModel(project_sig=mutator.project_sig,
1152                          app_name=mutator.app_label,
1153                          db_name=mutator.database)
1154
1155     evolver = mutator.evolver
1156
1157     if isinstance(field_type, models.ManyToManyField):
+ 752 lines
```

Figure 2.6: Code review in Review Board²

Review Board provides the following advantages and disadvantages:

Advantages:

- Review Board is Open Source under the MIT license [8]
- Reviews are not limited to code, images and files are also supported [57]
- Potential defects can be marked within an artifact if, for example, an image is under review [57]
- Interacts with Git, Subversion (SVN), Perforce, and so on [57]
- Reviews can be done before and after commits [57]
- Provides unified and side-by-side diff views [70]

²Source: www.reviewboard.org (last visited 30.09.2021)

Disadvantages:

- Reviews are based on manually generated diff-files [57], which is not the state of the art
- Frontend usability and appearance are outdated, as the combination of many different bright colors being used, the browser back button not being available sometimes and some buttons which are not recognizable instantly makes the tool nonintuitive to use in some areas.

2.5.2 Gerrit Code Review

Gerrit³ is an Open Source tool developed in Java which is available under the Apache 2.0 license. Originally it has been developed as a fork of Rietveld by Google for the Android project. [70] Gerrit can only be used before a commit, which means it only supports pre-commit code reviews. [52]

Once a developer decides to push his changes from the local commit to the Gerrit server the code review process is triggered. [52] The Gerrit server then automatically notifies the reviewers that there are new changes to the codebase which need to be reviewed before they are pushed to the version control system of choice. In this phase, the reviewers now have the chance to review the code and comment on the changes, as well as to vote in the end. [52] If the changes have not been accepted by the reviewers then the author of the code changes needs to improve the changes and fix reported problems. Otherwise, in case all reviewers approve the changes then the changes are sent to the version control system and are merged to the master branch. [52] The main focus of Gerrit is not just the improvement of code quality but also making sure that building pipelines run through. [44] In figure 2.7, the frontend of Gerrit is displayed, in which the code reviewing / code commenting process is shown. Gerrit provides the following advantages and disadvantages:

Advantages:

- Gerrit is Open Source under the Apache 2.0 license [70]
- Active development by Google [70]
- Provides unified and side-by-side diff views [70]
- Integrates well with continuous integration services like Jenkins [70]

³Gerrit Code Review: www.gerritcodereview.com (last visited 30.09.2021)

⁴Source: gerrit-review.googlesource.com/Documentation (last visited 30.09.2021)

2. RELATED WORK

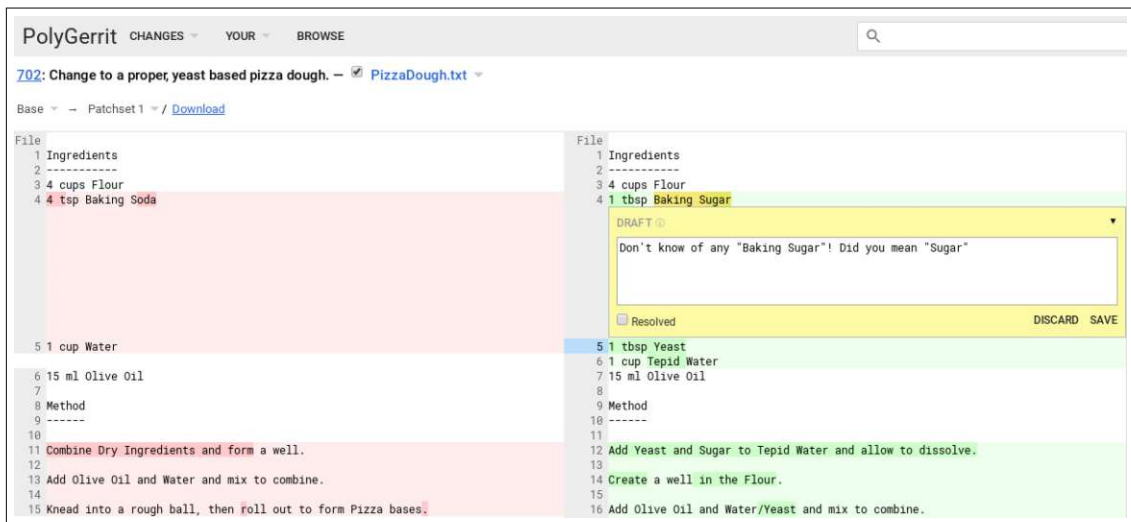


Figure 2.7: Code review in Gerrit⁴

Disadvantages:

- Can only review source code, other artifacts like images are not supported [70]
- Navigation within the frontend can be confusing [52]
- Technical problems like uncaught exceptions can appear while using Gerrit [52]
- Some error messages are uninformative and completely miss any details [52]
- Only works with Git [70]

2.5.3 Atlassian Crucible

Crucible⁵ is a code review application developed by the company Atlassian. [35] This allows Crucible to provide easy integration with other Atlassian products like Bitbucket, Jira, and other development tools of this company. [70] Crucible is a commercial tool and therefore has a subscription-based pricing model. It also does not provide any support for Open Source projects. [35] The main focus of Crucible is on post-commit reviews, but it also provides support for pre-commit code reviews. [70]

The workflow in Crucible is similar to the already mentioned code review tools. Once a component is ready for review the author can either select a group of reviewers or individual reviewers to check the components which are up for review. [70] The reviewers then have the chance to verify and comment on the code in the codebase and leave feedback. Crucible only supports unified diff views which cannot display the changes in a

⁵Atlassian Crucible: www.atlassian.com/de/software/crucible (last visited 30.09.2021)

side-by-side view. [70] After all reviewers have submitted their feedback, the author has to adapt the code according to the received comments. This iteration repeats until the reviewers accept the code. [70] Finally, when the reviewers are satisfied with the code they approve the component and close the review. In case of a pre-commit review, the merge can then be completed. [70] In figure 2.8, the frontend of Atlassian Crucible is displayed which shows the unified code diff view and some posted comments.

The screenshot displays a code review interface in Atlassian Crucible. On the left, a table lists reviewers and their changes:

Liang Zheng	acc14ed	59
Vitalii Petrychuk	1331923	60
Liang Zheng	acc14ed	61
Liang Zheng	acc14ed	62
Liang Zheng	acc14ed	63
Liang Zheng	acc14ed	64
Liang Zheng	acc14ed	65
Lukasz Kuzynski	14e462d	64
Lukasz Kuzynski	14e462d	65
Lukasz Kuzynski	14e462d	66

The main area shows a unified diff view of JavaScript code. The code is color-coded: red for deletions and green for additions. The code includes a function to convert strings to numbers and a function to find ranges in an array.

```

var line = lines[i];
if (count !== 0 && line === (parseInt(lines[i - 1], 10) + 1)) {
  if (line !== (parseInt(lines[i + 1], 10) - 1)) {
    lineStr += "-" + line;
    count = 0;
  } else {
    count++;
  }
}

var ranges = [];
lines = lines.map(function (line) {
  return +line;
});
var rangeStart;
var rangeEnd;
for (var i = 0; i < lines.length; i++) {
  rangeStart = lines[i];
  rangeEnd = rangeStart;
  while (lines[i + 1] === lines[i] + 1) {
    rangeEnd = lines[i + 1];
    i++;
  }
}

```

Below the code, there are two comments:

- Cezary Zawadka**: I guess it's JS way of converting strings to numbers?
Reply · Mark as unread · Add to favourites · Create issue · 15 Sep
- Lukasz Kuzynski**: Yep - the shortest way 😊
Reply · Leave unread · Create issue · 15 Sep

Figure 2.8: Code review in Atlassian Crucible⁶

Atlassian Crucible provides the following advantages and disadvantages:

Advantages:

- Interacts well with other Atlassian products like Jira, Bitbucket, etc. [70]
- Provides a timeline to view review iterations [70]
- Interacts with Git, Subversion (SVN), CVS, Perforce [70]
- Provides a frontend with a modern appearance

⁶Source: betterfasterhappier.medium.com (last visited 30.09.2021)

Disadvantages:

- No side-by-side diff view available [70]
- Closed source, code not available for the community [35]
- Commercial tool with high license costs [35]
- Only source code reviews are possible, images are for example not supported [70]

2.5.4 GitHub

GitHub⁷ is an online hosting service for git repositories. In general, GitHub provides a free version with limited features available and a subscription-based enterprise version. [70] It is not possible to deploy GitHub on a private server unlike the other tools already presented GitHub is an online service and therefore can only be accessed via the official GitHub servers in the cloud. [70] This fact does not only bring disadvantages with it but also some advantages for example that there is a big amount of skilled people available on GitHub who can review code and request changes in Open Source projects. It would not be possible to have so many people available on a self-hosted code review instance. [80]

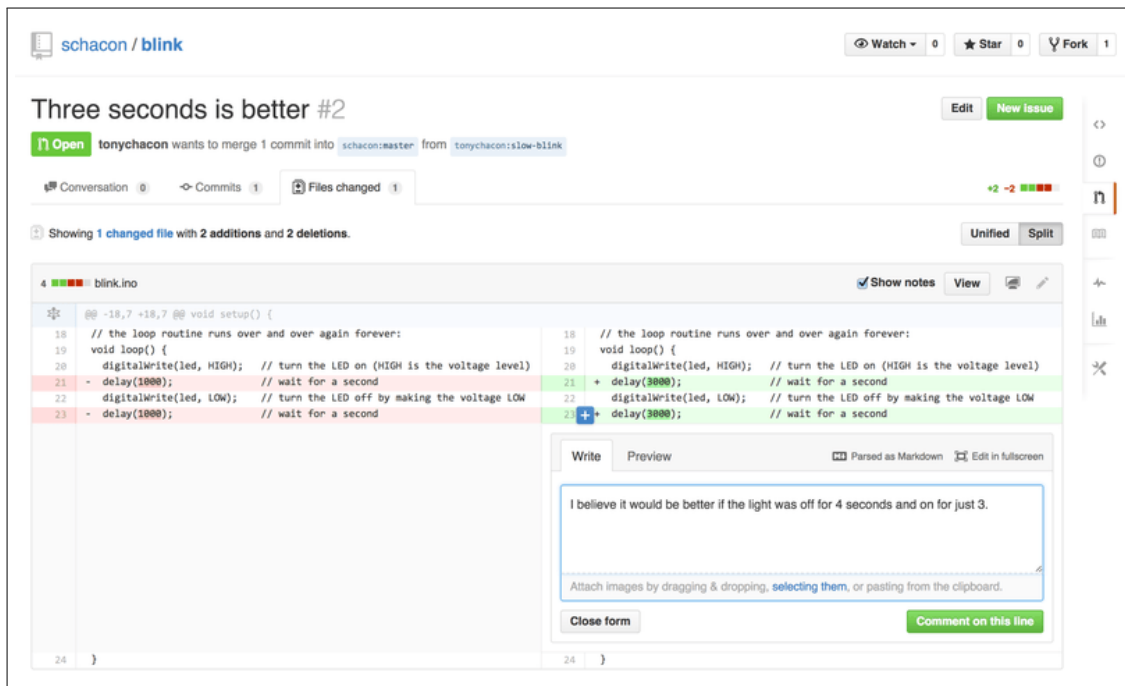
GitHub offers a lot of features as it is a full git repository service, but especially its review component is used by many developers. The review functionality of GitHub is called Pull Requests. [80] Pull Requests are a simple concept that provides code review functionality on diff chunks, which are views of snippets of code that has been added, altered, or deleted. [35] Even though Pull Requests do not provide advanced review functionalities they are widely used on GitHub. [35] The participants in the review can comment on the code similar to the other tools that have already been mentioned but also a basic voting functionality is available in GitHub Pull Requests. Once the author has fixed all problems reported by the reviewers and the reviewers also accepted the Pull Request it is automatically merged into the destination code branch. [35] In figure 2.9, the frontend of GitHub is displayed that shows the side-by-side diff view within the code reviewing process including the comment section. GitHub provides the following advantages and disadvantages:

Advantages:

- Provides unified and side-by-side code diff views [70]
- Easy start and no setup required because the service is hosted in the cloud [70]
- Modern frontend appearance and good usability
- Fully integrates Git in the workflows but also SVN access is available [70]

⁷GitHub: www.github.com (last visited 30.09.2021)

⁸Source: git-scm.com/book/de/v2/ (last visited 30.09.2021)

Figure 2.9: Pull requests in GitHub⁸

Disadvantages:

- Closed source, code not available to the community [70]
- Only limited support for reviewing images available [70]
- All data is stored in the cloud, no self-hosting available [70]
- GitHub Pull Requests are widely used but only a relatively simple form of reviews [35]
- Only some features are available for free [70]

2.5.5 Summary

In this section, we presented some well-known software inspection tools and mentioned different advantages and disadvantages. Table 2.1 presents an overview of important features and characteristics and compares the mentioned software inspection tools to each other.

Especially, the features *model/image review*, *review administration*, and *experiment administration* are the central point of focus. The inspection tool Review Board provides the most advantages within our use case in comparison to all other presented tools.

2. RELATED WORK

Feature / Characteristic	Review Board	Gerrit	Crucible	GitHub
Open Source	✓	✓	–	–
Non commercial (free)	✓	✓	–	✓
Code review	✓	✓	✓	✓
Model/image review	✓	–	–	–
Review administration	✓	✓	✓	✓
CVS support eg. Git, SVN	✓	✓	✓	✓
Model design	–	–	–	–
Experiment administration	–	–	–	–

Table 2.1: Comparison of advantages and disadvantages of the inspection tools

However, there are still features missing that are relevant for the research questions which are presented in section 4. For example, RQ2 focuses on the development of a prototype for tool-supported review and experiment administration of which experiment administration is not available.

In summary, it can be seen in table 2.1 that there is currently no single tool available that supports all focused features. Therefore, the model design and review editor (MDRE) has been implemented and is presented in detail within section 5. This tool focuses on the implementation of new features in the field of software inspection tools and has been advanced to an environment that is capable of supporting all features that are required to solve the research questions.

Experimentation

In this chapter we explain the most important details about the topic of experimentation. Within this work, experiments are in the central focus to solve the research questions and therefore are needed for the understanding of later chapters.

In every mature and advanced scientific topic, there is the need to understand the given components and the relationships between the components. [5] In the field of software engineering researchers nowadays develop more practical solutions during their research than ever before. The aim of their research is that practitioners should use their developed software systems or research prototypes that they developed in the research laboratories with the goal of the innovation of software products and processes. [42]

In the scientific software engineering community, there is a big amount of research going on with many researchers all over the world searching for new solutions and innovations. Researchers tend to only point out the technical benefits of their newest inventions, without providing pieces of evidence on the usefulness of their new tool or method in practice. [42]

Furthermore, the available technical and practical information about a newly invented solution is not always supported by evidence that would be needed to assure the usefulness in real processes or the applicability in business processes. Therefore, it can be hard for practitioners to find those research solutions which achieve the best innovations and improvements in practice and to the current state of the art. [42]

Scientific knowledge needs the validation of theories, principles, and practices to show its correctness. A possible solution to produce evidence for scientific knowledge is empirical studies. [42] In scientific research the term validity has many interpretations, most of them focus on checking if the measure conforms to what it was designed to represent. [14] In general, there are three different types of validity identified:

- **Content validity:** The content validity depends on the carefulness with which the desired domain has been covered. [14]
- **Predictive validity:** This type of validity involves predicting the outcome of an event with the help of the measurements. [14]
- **Construct validity:** In this validity type the focus lies on finding out how close an operational definition results in data that is related to an abstract construct. [14]

For conducting a validation there are two options, either the proposal has to be proven formally, or the empirical methods have to be used to collect evidence for the research results. Formal methods are not common in scientific software engineering research, because it is often simply not possible. This has lead researchers to focus on empirical methods to find evidences for their study results. [36] The different empirical methods are either quantitative or qualitative evaluations. Quantitative methods work by measuring certain effects, while qualitative methods are used to search for relations and reasons behind the observed behavior. [36] Some examples of quantitative evaluations are experiments, case studies, surveys, field studies, and meta-studies. A few examples of qualitative evaluations are interviews and group discussions. [36]

Nowadays, most scientific research is supported by evidence which results from experimentation. During the experimentation, the research purposes and measurements of the variables involved in the phenomena are observed. [32] There are multiple common excuses why empirical studies and especially experimentation are not used to evaluate the found research results: [32] [36]

- Traditional scientific methods cannot be applied.
- The current level of experimentation is good enough.
- Experiments cost too much.
- Demonstrations suffice.
- You never get the results published.

All of these fallacies and some additional ones have been shown as exaggerated concerns by Juristo et. al. in [32]. In reality, experimentation provides the basis for the improvement in knowledge and understanding within a scientific topic. Software Engineering is one of the fields which is a good candidate for using experimentation to gather additional knowledge and understanding. [5] Experimentation can help to better evaluate, predict, understand, control, and improve the software engineering process and the systems developed within it. [5]

The controlled experiment is also the method that provides the highest degree of confidence about the results. In an experiment, researchers try to control every variable that can influence the outcome except the variables which should be analyzed. [36]

In summary, empirical strategies rely on evaluated results that have been observed or collected during experimentation. Experiments are an empirical strategy in software engineering. [78] Experimentation provides a systematic, disciplined, and controlled way for evaluating activities performed by humans. There are also other empirical strategies for software engineering like surveys and case studies. [78]

No matter in which project or scientific field if we want to investigate the effectiveness of a tool, technique, or method scientifically and not just rely on the "common wisdom" we need to perform an empirical study. [54] Therefore, in the following sections, the most important principles for scientific investigations are presented.

3.1 Principles of Investigation

There are certain principles for the investigation during an empirical study that we need to follow, to get results that can be verified and validated scientifically. [53]

3.1.1 Hypothesis

In the first step, the exact goal of what should be investigated within the study is formulated. The initial formulation helps to choose the best research technique in a later step. This formulation about the goals of the research is called the hypothesis and states what we want to know after conducting the research. [53] A hypothesis is a tentative theory that is used to explain the behavior which should be explored. The hypothesis of an empirical study is expressed in two different statements: [54]:

- **Null hypothesis:** The null hypothesis assumes that there is no significant difference between two treatments, methods, tools, techniques, environments, or other conditions which are measured, used on the same dependent variable. [54] Pfeeger et. al. state the following example for a null hypothesis in [54]:

There is no difference in the number of defects per thousand lines of code between code tested using coverage measures and code tested using an operational profile.

- **Alternative hypothesis:** An alternative hypothesis states that there is a significant difference between two or more treatments [54]: Pfeeger et. al. state the following example for an alternative hypothesis in [54]:

Code tested using coverage measures has a significantly different number of defects per thousand lines of code than code tested using an operational profile.

As stated above the null hypothesis always states that there is no difference between the different treatments. It is always assumed that the null hypothesis is correct unless

data can be generated during the research which can prove the difference. [54] Therefore, every investigation within an empirical study should focus on starting from the null hypothesis and test this hypothesis. In case there are results found during the study which are convincing enough then the null hypothesis can be rejected. [54]

3.1.2 Investigation technique

After the hypothesis has been defined the investigation technique which either confirms or refutes the null hypothesis must be chosen. There exist three different types of investigation techniques:

- **Survey:** A survey documents the relationships and the outcomes of a situation in the retrospective. Therefore, surveys are always performed after a certain event has finished. [54] A survey has a wide coverage and so usually provides a clear overview of the area of interest. [29] During the survey data is evaluated which has been collected during an event by others or which has been recreated with the help of the recollection of different events. [54] Scientists often ask people questions during surveys to extract data about certain events. With the help of this information, the researchers can find out how the products, processes, and resources were affected by each method, tool, or technique. Another possibility within surveys is to determine trends and relationships among the research results. [54] The biggest benefit about surveys is that no control over a certain event or situation which is investigated is needed. [54] Surveys tend to determine what is happening generally over large groups in projects.
- **Case Study:** In a case study every aspect that should be investigated must be decided in advance. During the activity which should be investigated the specified data is captured and analyzed afterward. In a case study, for example, the first key factors are identified which could affect the outputs in some way. [54] Then, the activity's inputs, constraints, resources, and outputs are observed and so the specified factors and how they affect the outcomes can be documented in this step. In a case study researchers try to avoid small scales and laboratory situations by investigating real and typical projects instead of trying to capture every information and all possible combinations. [54]
- **Controlled Experiment:** Similar to a case study also in an experiment every aspect that should be investigated must be decided in advance. [54] The main difference between an experiment to a case study is that the experiment requires a great deal of control. [66] Experiments have the purpose to prove or disprove a causal relationship between an observed attribute and the outcome of the activity performed. [29] In an experiment, every variable within the activity which could affect the output is under control. Since this is hard to achieve and laborious, experiments are usually smaller than case studies and executed with only a small number of experiment participants involved. During an experiment, a small set of aspects from a larger problem is examined. [54]

3.1.3 Control over variables

After the hypothesis is specified it must be decided which variables might affect the outcomes of the study. For every variable, the degree of control that is possible within the study must be determined. [53] The control of the variables is an important factor when choosing the right technique for investigation. [66] As already presented earlier, an experiment requires a great deal of control, while a case study is more efficient when some or all of the relevant variables cannot be controlled. [53] For example, suppose that we want to know whether a certain development method produces higher quality program code than another. In case we cannot control the development method which each participant is using then a case study is needed to collect the desired results. [53] On the other hand, if it is possible to decide which participants are using a certain development method then an experiment can be used to elicit the results. [53]

3.1.4 Meaningfulness

To perform meaningful investigations four different aspects need to be considered: [54]

- **Confirming Theories:** Often tools and certain techniques are adopted because "conventional wisdom" or "expert judgment" suggests that these are the best options and even whole standards can be derived in this way. The problem with this approach is that there is little to no quantitative evidence that can support the effectiveness or utility of the techniques, tools, and standards. [54] To provide scientifically accepted evidence, case studies and surveys can be used to evaluate certain claims in small environments like single organizations, and on the other hand, experiments are used to evaluate these claims in a general way which are also valid outside of a single environment. This evidence is valid for every possible environment, for example, many different organizations that are not related to each other. [54]
- **Exploring Relationships:** During an empirical study it is not just the goal to find out if some tool, technique, or method is working or not, it is often also important to investigate if which relationships between various aspects exist. [53] The understanding of relationships is the main goal for the success of every project. Therefore, in the investigation, the expected relationships are expressed in a hypothesis and an experiment can be executed to test in which form the relationship holds. [53]
- **Evaluating the Accuracy of Models:** In some cases models are used to predict the outcome of an activity or to describe the usage of a tool, technique, or method. [54] An investigation can reveal the accuracy and dependability of the model which is achieved by comparing the predictions with the actual results. Often the design of studies about models must be difficult because the predictions can manipulate the outcomes. [54] The main reason for this is that usually, the predictions become goals and so the participants may unintentionally change their behavior in favor of

the goals which have been set. Therefore, when designing a study the participants should not know the predictions until the study is fully completed. [54]

- **Validating Measures:** During an investigation measures capture certain attributes of products, processes, or resources. Measurements are valid if they match the characteristics of an attribute under changing conditions. During every investigation, it should be made sure that a measure always represents the attribute it claims to quantify. [54] This can be proven with the execution of a study that validates if the measure changes appropriately as the measured attribute changes. The study must include a model which describes how the measures relate to the real objects and activities. [54]

3.2 Attributes & Types of Experiments

Before an experiment is executed all parameters and the whole environment are described and fixed to the defined values. This gives experiments the advantage that the experimenters have full control over the whole scenario and can manipulate the behavior directly, precisely, and systematically. In an experiment, it is also possible to make the participants use different methods to achieve the same goal. [78] Four different attributes can be adjusted during the experiment planning or design: [4] [36] An experiment can present the following results: [4]

- **Descriptive:** The relationships between variables have not been examined. Although, there could be patterns within the variables, which can be examined later. [4]
- **Correlational:** The variation of dependent variables is related to the variation of independent variables. [36]
- **Cause-effect:** Independent variables are the only cause for a possible variation independent variables. [36]

Every experiment can be performed with the following two different types of participants or an even distribution of both groups: [4]

- **Novice:** The whole experiment is executed by only inexperienced participants, for example, students. [4]
- **Experts:** The participants in the experiment have experience in the study domain. [36]

Experiments can be performed in two different environments: [4]

- **In vivo:** The experiment is executed in the field, which means in a real software industry environment. This makes sure that the whole experiment is executed under realistic conditions. [36]
- **In vitro:** The execution of the experiment is done fully in a laboratory under completely controlled conditions. [4] An example of an experiment executed in vitro is a study executed at a university. [36]

Experiments can be performed with a different level of control: [36]

- **Controlled experiment:** These experiments are typically performed in vitro with inexperienced participants like students at a university. This type provides strong statistical confidence in the results, is expensive to perform, and is difficult to control. [36]
- **Quasi-experiment:** Within experiments with this level of control the execution is usually done with experienced participants. This experiment type is typically performed in vivo with a qualitative character. [36]
- **Observational study:** During an observational study there is no treatment or control of the variables involved. There usually is no set of study variables that is defined before the experiment is executed. [36] [64]

In general, there are not just attributes that can be adjusted in multiple experiments but there are also two different types that an experiment can focus on:

- **Human-oriented experiment:** In a human-oriented experiment the participants use different methods to achieve the goal, for example, there could be two different inspection methods applied to test the same piece of software code.[78]
- **Technology-oriented experiment:** On the other hand in technology-oriented experiments, the same methods are applied but there are different tools used to achieve the goal of the experiment. This could for example be the usage of two different test case generators which are applied to test the same code snippets. [78]

3.3 Experiment design & execution

Experiments are preparation-intensive tasks to execute. Therefore, it is important to ensure that the goals of the experiments can be fulfilled, to make sure that the time and resources put into the experiment pay off. For this purpose, the exact procedure in planning and executing an experiment is defined by Wohlin et. al. in [78].

The process of an experiment has five different phases which are executed. [59] Figure 3.1 displays an overview of the experiment phases in logical order.

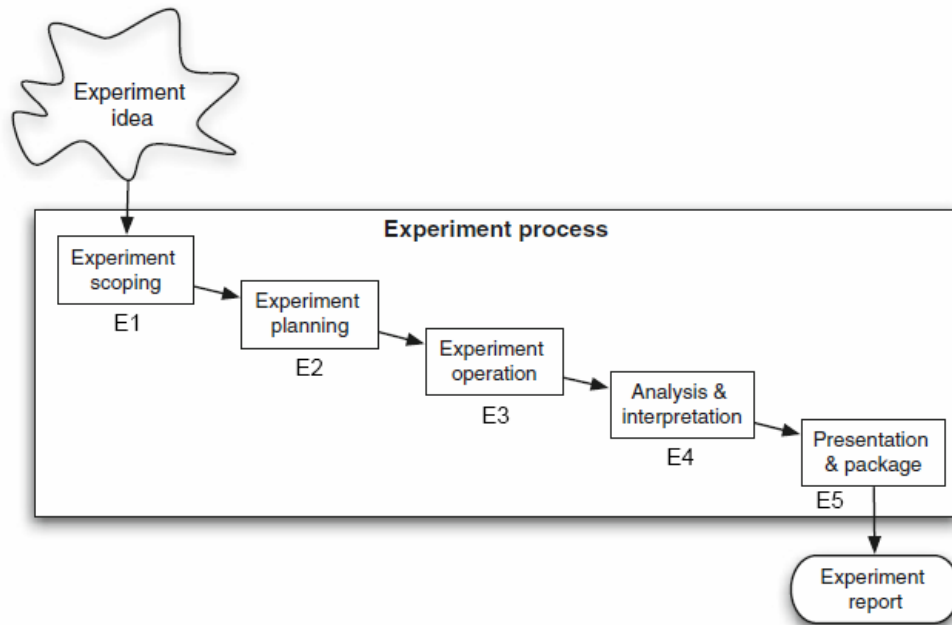


Figure 3.1: Phases of an experiment [78]

1. **E1 - Scoping:** In the first phase the experimenters define the experiment goals, scope, object of study, and hypotheses of the experiment. [59] The foundation of an experiment is laid by defining the goal of the experiment. An object of study is the entity that is studied during the experiment. It can be a product, process, resource, model, metric, or theory. [78]
2. **E2 - Planning:** After the scoping determined the foundation for the experiment the experimenters plan the context of the experiment. This includes human resources, the environment, experiment design, and hypotheses that are based on the experiment goal. [59] Developing an exact plan for the experiment is important to control the experiment. If an experiment is not planned properly the results of the experiment can be disturbed or even be destroyed and so become worthless. [78]
3. **E3 - Operation:** After an experiment has been scoped and planned it must be carried out to be able to collect all the desired data which result from the experiment. [78] During the operation of the experiment, the experimenters prepare all materials necessary and execute the experiment according to the previously performed planning and scoping. [59] The operation phase consists out of three steps, the first step is the preparation in which the participants are chosen and

forms, guidelines, etc. are prepared. [78] During the next step, the execution the participants perform the tasks according to the guidelines, and data is collected. And in the last step, the data validation is performed and all the results which have been collected are validated. [78]

4. **E4 - Analysis and Interpretation:** The data collected during the experiment is the input for the analysis. [59] The main goal of the analysis phase is to interpret the results, to be able to draw conclusions based on this data. [78] Therefore, the data is analyzed by the experimenters using methods like descriptive statistics, data reduction, hypothesis testing, etc. [59]
5. **E5 - Presentation and Package:** After the completion of an experiment the findings can be presented to different audiences. [78] All data collected during the experiment is interpreted and a conclusion is drawn. The results are usually documented and reported within a research paper or packaged for replication. [59] There is also the possibility of publishing a technical report parallel to a possible conference paper in case there is not enough space left in the paper to publish the whole results. [78]

3.3.1 Realism in Experiments

An important criterion for successful experiments in any applied scientific field is the possibility to adopt the research results into practical industrial engineering. Therefore, it is important to achieve high homophily. Homophily is the degree to which the innovator from the scientific field and the possible adopters from the industrial practice conform to certain attributes like objectives, beliefs, norms, experience, and culture. [65] The opposite of homophily is called heterophily which is one of the biggest problems in the implementation of innovations. If the possible adopters and the scientists are heterophyllous then a possible adoption of the innovation into the practice is not realistic because the experiment differs too much from the reality. [65]

Therefore, the experimental environment and the real industrial environment must be close to each other, which encourages the practitioners' attention to the innovation. [65] So far most studies that have realized realism are case studies. A drawback of case studies is, as already explained, that many variables vary from one case study to another and so the case study is often only valid for a single environment. [54] To overcome this shortcoming, there exists the possibility to conduct a controlled experiment as a complement case study. [65]

An experiment needs to have two different types of realism. The first type is experimental realism, which refers to the impact of the experiment on the participants. Experimental realism is present if the experiment appears to be real and meaningful to the participants and the encountered perceptions in the experiment match the real world. [65] The second realism type is called mundane realism, which refers to the relation between the experiment situation and the real world. This means the ability to generalize results

from the experiment environment to the real world in industrial practice is an important realism attribute for experiments. An experiment is realistic if the situation to the participants is realistic and the participants react to the situation in the same way as they would react to a situation that happens in real life during a random activity in an industrial environment. [65] There are three main challenges when aiming for realism in experiments:

- **Realistic tasks:** An important challenge is the size, complexity, and duration of the executed experiment tasks. [65] Many experiments simplify the whole process a lot in which experiments that are executed in only a few hours are expected to return the same results as an observation for various months which is unrealistic. Meaningful results need to be observed over longer time periods and can often not be achieved within an experiment that only lasts a few hours. [65]
- **Realistic subjects:** To achieve results that can be generalized for industrial environments the selection of the participants who perform the experiment tasks is an important challenge. Since it is still unclear how well an experiment with only students participating can be generalized to professional environments, it can be an advantage and important to use both inexperienced and experienced participants in an experiment. [65]
- **Realistic environment:** The last challenge is that the tasks should be carried out in a realistic environment. [65] Performing experiments with realistic tasks and participants in an unrealistic environment is a common pitfall in experiments. To generate meaningful results, the experiment must be executed in an environment with an infrastructure of supporting technology that includes processes, methods, tools, and so on. [65]

3.3.2 Experiment Reporting

The number of individual empirical studies of technologies reported in scientific publications like journals at certain conferences is increasing every year. But at the same time, there are only a few success stories in which technologies were adopted to the industrial practice because of the evidence provided with the help of an empirical study. [27] This problem along with all the loss of potential has for example been reported by NASA. [27]

Another example is the area of software inspections for which success stories and empirical evidence showing their usefulness have been available for many years but they are not widely used in industrial practice. This raised the question if the wrong type of evidence, the wrong format, delivery to the wrong people, or communication of the evidence via the wrong channels could be the key problems for this phenomenon. [27]

One of the main problems with integrating study results into the general knowledge and also into the industrial practice is the heterogeneity of the study reporting. It is often difficult to find the required information because the information is duplicated

between individual sections of the study reports and the important information that matters is often not present in the reports. [26] For example, in many study reports, the information about the general context is reported differently while a generalization is not taken into account. At the same time, specific information, which would be helpful for the practitioners to find out about the overall impact of the technology on a project or business domain, is often missing. [26]

A possible solution to avoid the heterogeneity of the reports is to define and introduce guidelines for reporting empirical studies like experiments. [26] According to Jedlitschka et. al. in [26] a useful experiment reporting guidelines should include the following elements: *Title of the study, authorship, structured abstract, keywords, introduction to the topic, background, experiment planning, experiment execution, analysis, discussion, conclusion, future work, acknowledgments, references, and appendices.* [26]

In summary, the execution and administration of experiments are usually performed manually without the help of any additional tool support. Therefore, conducting experiments is laborious and requires a lot of effort. To solve this problem the implementation of tool support for the execution and administration of experiments is an important requirement.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Research Questions & Solution Approach

This chapter presents the methodology, all research questions which are answered, as well as the solution approach which is used to find the answers to the research questions.

We want to improve the review cycle for reviewing models by implementing tool support that provides the possibility to report defects in an artifact. Additionally, we also aim to implement tool support for review and experiment administration. With the help of such a tool support component, review methods can be compared to each other during experiments using a qualitative or quantitative comparison. The concrete research questions are presented in the next sections.

4.1 Design Science

We use design science as a research methodology to achieve and evaluate all goals of this thesis. Design science is used in many different engineering disciplines [18] worldwide and aims to change existing situations within the organizational context into preferred ones during an engineering process. [43] This means that real-world problems are solved by researchers who develop general knowledge in a certain area to help practitioners create solutions in their problem contexts. [18] In this research methodology, innovations are created that define ideas, practices, technical capabilities, and products. [24] With the help of these innovations the analysis, design, implementation, management, and use of new systems can be carried out effectively and efficiently. [24] [43]

There are two major components within Design Science the object of study and the context in which the artifact resides. The selected artifacts are designed and investigated during the design science methodology. [51] Every artifact needs to be designed to interact with a certain problem context and improve something in this chosen context. There are

4. RESEARCH QUESTIONS & SOLUTION APPROACH

a lot of entities that can be designed as an artifact like software and hardware systems, organizations, business processes, services, methods, techniques, models, instantiations, social innovations, and so on. [75] [51] For example, in software engineering, the practical problems within the context are mostly problems in the design, implementation, or maintenance of a software artifact. [74] Additionally, some entities cannot be artifacts that are being studied with the help of design science like people, values, desires, fears, goals, norms, and budgets. These elements appear in the context of artifacts but cannot be designed by the researchers, therefore they cannot be artifacts themselves. [75] Design science differs from the professional design of a solution in the way that in the design science research new and for a community interesting knowledge is produced. [71] On the other hand, in the typical industry design use case, only a new artifact is produced but there is no focus on producing new knowledge. [71] The reason for this is that in professional design existing knowledge is used to solve organizational problems using artifacts that exist in the knowledge base. [23]

The produced artifact itself does not solve or improve any problems but the interaction between the artifact and the problem context does solve certain problems. Artifacts can interact differently with individual problem contexts, therefore they can solve different problems in each context. [75] This means that it is important in design science that the interactions between artifacts and problem contexts are studied by the researchers and not the artifact itself. [75] The design science research cycles are displayed in figure 4.1.

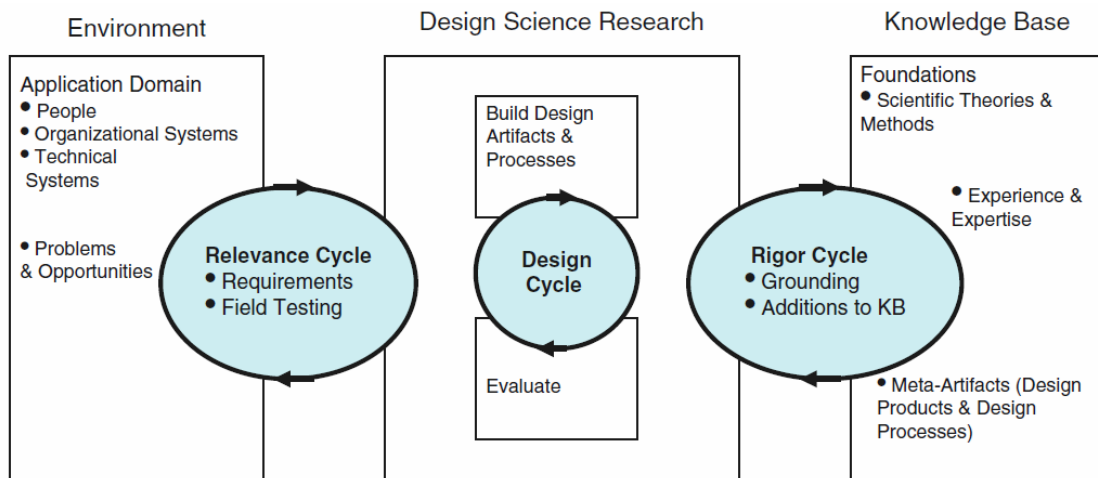


Figure 4.1: Design science research cycles [23]

Every design science research project consists out of three different areas namely the environment, the design activities, and the knowledge base. [23]

- **Environment:** The environment contains the problem which is observed during the design science research. This means that the phenomenon of interest is observed by the researchers within this environment. [17]

- **Design science research:** During the design science research an artifact is developed which can solve problems by interacting with a problem context. This procedure has already been explained in detail previously within this section.
- **Knowledge base:** The knowledge base is the environment in which all previously documented and developed theories, methods and artifacts can be retrieved by the researchers. [17] The knowledge base itself is usually insufficient for the development of new artifacts. Therefore, many researchers use their own experience or go by trial and error when designing new artifacts. [17]

In addition to the three areas, there are also three different design science research cycles, which represent the relationships between the mentioned areas. [23]

- **Relevance cycle:** The relevance cycle defines the application context for the design science research. It provides not just the requirements for the artifact which is designed but also defines the acceptance criteria for the evaluation and research results. [23] The iterations of the relevance cycle depend on the results from the field testing. There are additional iterations needed if for example incorrect or incomplete results are collected during the field tests. [23]
- **Design cycle:** The internal design cycle is the most important part of every design science research project. Within this cycle each iteration loops between the design and the evaluation of an artifact. This makes it possible to collect feedback early in the development process and allows detecting and fixing problems during the design. [23]
- **Rigor cycle:** The rigor cycle provides knowledge to the design research project which has been published in the past and counts as state of the art. [23] An important aspect in design science research is that the artifact must not be based on some kind of already known design process which would result in the artifact being a copy of an already known design. Therefore, the rigor cycle makes sure that the researchers can research and reference the current knowledge base to guarantee that the artifact produced is a real research contribution. [23]

4.2 Prototype for tool-supported model reviewing (RQ1)

Due to missing tool support, performing different review strategies is time-consuming and generates an administrative overhead, as materials are provided and results are collected using different platforms. Furthermore, the review strategy is often performed in a traditional way using pen and paper. With the help of tool-supported model reviewing, materials are provided using the same digital platform in which the review is performed, reducing administration and time overhead. Therefore, we define the research question:

RQ1: How can model reviews be supported with the help of a tool support prototype?

To address RQ1, we implement a prototype for tool-supported model reviewing as the foundation to increase the effectiveness of reviews. Furthermore, we make sure that reviews can be linked to a tool-supported review administration component for a more efficient coordination.

Our goal is the implementation of process support for different review approaches. To achieve these goals an extension for the *Model Design and Review Editor* (MDRE) is implemented. We design a process support system for inspections and reviews with suitable methods and guidelines. Therefore, the added process support implementation supports performing reviews in the three stages of planning, execution, and reporting of the individual reviews. The focus of the supported reviews is the analysis and improvement of the quality of each artifact that is under review. With the help of the implemented process support, reviewers can execute the review technique which they were assigned to easier, faster, and more reliable, as there is a component in place which avoids errors and makes sure that the guidelines for the selected review type are not violated. This is also an advantage for the review managers who are in charge of the review itself, as there are strict guidelines that cannot be violated and so the results of the review are less likely to be wrong. Once this research question is solved the review component of the MDRE offers the possibility for reviewers to review artifacts within the MDRE and report defects within them.

4.3 Review and Experimentation Process Support (RQ2)

Planning and administration of experiments are laborious since in an experiment many different experimenters and participants need to be supported. In an experiment materials need to be provided, prerequisites and final tasks need to be performed before and after every experiment and the experiment itself needs to be managed. The generated administration effort is hard to overcome without any provided tool support which assists in managing experiments. With the help of tool support for experiment administration, the effectiveness of experiments is increased and the effort is reduced. Therefore, we define the research question:

RQ2: How can the administration of reviews and experiments be supported with the help of a tool support prototype?

To address RQ2, we design a method for tool-supported review and experiment administration. For the comparison of two or more review approaches to each other the implementation of a review and experiment administration component is added to the MDRE. This component supports the planning and the execution of an experiment including the reviews which are executed within the experiment. In the planning stage, the experiment is set up and prepared for mostly automated execution. Therefore, participants can be imported into the MDRE and are assigned to groups of participants with different review techniques assigned to them. For example, half of all groups perform a pen-and-paper-based traditional review, and the other half uses the MDRE approach. The same base model is used within one group type to make sure that the reviews

carried out can effectively be compared to each other. Before the participants are allowed to execute their tasks, they need to fulfill all prerequisites. For example, the answer of an experience questionnaire, signing a data protection form, and completing other experiment-related tasks, which are needed to be done by the participants before the experiment starts. Once all prerequisites are fulfilled they can start with their tasks, additional materials, and a task description can also be provided by the experimenters for each review type in the experiment individually. The process support for the different review types assists the participants during the experiment execution.

The implemented component supports the evaluation of the review methods by providing metrics about the procedures and the results of the individual reviews. Part of these metrics is for example the total amount of errors that were found, the time until the first error was found, location in the model where the errors were found, and so on. These can later be used by scientists to compare review methods with each other and find out the pros and cons of each method. Metrics also help to find out which one is performing better when multiple review methods are used on the same base model. After all tasks were successfully carried out by the individual groups the collected results can be evaluated by the experimenters and a conclusion about the experiment is drawn. The group members finish the experiment by accomplishing some smaller final tasks, for example, answer a feedback questionnaire, hand in every hardcopy which has been provided at the beginning, etc. All tasks, which need to be carried out, can be monitored via a checklist in the MDRE.

4.4 Benefits and limitations of MDRE (RQ3)

Evaluating the benefits and limitations of a newly proposed solution is an important part of scientific research. Practitioners from the industrial field need the evaluation of the scientific results to make sure that there are benefits that they can profit from when they introduce the proposed solution. But also other scientific researchers are interested in the evaluated results to be able to verify the results of the research study. This means that it is important that the results are evaluated to show the limitations and benefits of the implemented tool support. Therefore, we define the research question:

RQ3: To what extent are the model review and experiment administration processes improved by the two implemented tool support prototypes?

To address RQ3, we plan, design, and execute a controlled experiment in a classroom setting. The planning of the experiment is done with the help of the implemented experiment and review administration tool supports, which have been integrated into the MDRE. During the experiment – along with the main task of reviewing some artifacts – the participants also have to provide additional information like their experience and answers to feedback questionnaires, which are collected and used in the evaluation of our research questions.

In the experiment itself, the participants are split up into two different groups. The first

4. RESEARCH QUESTIONS & SOLUTION APPROACH

group uses a traditional pen-and-paper-based guided review, the second group uses the MDRE approach for inspecting the provided artifacts. At half time of the experiment, the groups are swapped to make sure that valid and comparable data is produced during the experiment.

For the evaluation of RQ1, we collect feedback about the improvement which is achieved by the newly implemented tool support for the different review strategies. The data is collected with the help of a feedback questionnaire by asking all review participants, who used a tool-supported review strategy during the experiment, for their feedback on the new implementation. Furthermore, the metrics that have been collected during the reporting of the defects are compared. For example, measures like time until the first defect has been found, the number of defects found per time unit, total defects found, etc. can give useful evidence when the two group types are compared.

In the evaluation of the second research question RQ2 the focus is on evaluating the resulting improvement in efficiency and administration effort provided by the implemented experiment administration tool support. An improvement has been achieved if the data is no longer distributed over many different locations and files but collected in one centralized place which is the MDRE and so makes experiment data easier to setup and manage. Furthermore, another improvement would be that the preparation before and the cleanup after the experiment can be done quicker than without the experiment administration tool support.

Model Design & Review Editor

This chapter presents the *Model Design and Review Editor* (MDRE) that we use as the base application to solve the research questions of this work. We advanced the MDRE with two different AddOns which are also explained in the next sections.

In section 2.5, we presented existing and well-known software inspection tools. These software inspection tools provide a variety of features to support the software inspection process. Table 5.1 provides an overview of the features from the software inspection tools and the aimed features of the (MDRE) at the end of this work. All important features that are required to fulfill the research questions are marked in blue. However, as it is shown in the comparison within table 5.1 there is currently no application available that is capable of all features required for solving the research questions. Therefore, in the practical part of this work, we use and improve the MDRE. Our main focus is the design and implementation of a prototype for improved model review and experimentation

Feature / Characteristic	Review Board	Gerrit	Crucible	GitHub	MDRE
Open Source	✓	✓	–	–	planned
Non commercial (free)	✓	✓	–	✓	planned
Code review	✓	✓	✓	✓	–
CVS support e.g., Git	✓	✓	✓	✓	–
Model design	–	–	–	–	✓
Model / image review	✓	–	–	–	✓
Review administration	✓	✓	✓	✓	✓
Experiment administration	–	–	–	–	✓

Table 5.1: Comparison of the MDRE with well-known software inspection tools. The blue features are important for solving the research questions. [56]

administration tool support into the MDRE. To achieve this goal the initial state of the MDRE has to be advanced to support the required functionality. But also with the development of additional features into the MDRE, it is not capable of all features provided by the combination of the presented software inspection tools. However, this is also not necessary because the MDRE is not meant to be a universal tool supporting the whole software inspection process in one application. For example, the MDRE was never intended to become tool support for code reviews and therefore also does not need any CVS integration within it. It is currently sufficient for the MDRE to only support the highlighted features that are focused on within this thesis.

To evaluate the results that have been achieved we conduct a controlled experiment with about 80 participants who use the MDRE to evaluate the newly implemented prototype tool supports. The evaluation is explained in detail in the next chapter. Within this section, the initial state, as well as all improvements of the MDRE that have been developed, are explained in detail.

5.1 Initial State of the MDRE

In this section, we present the initial state of the MDRE at the beginning of this work.

5.1.1 Introduction

The MDRE is an application that was initially developed at the Technical University Vienna between February and August 2020. The main reason for the development of the MDRE was the fact that in software engineering there is a variety of tools available that aim to provide support for reviews on different artifacts like models, source code, images, and documents. [77] However, most of the available review tools do not provide support for designing and reviewing artifacts at the same time. Usually, there are different tools needed for designing an artifact and later reviewing it which leads to increased overhead and complexity of the whole model engineering process. [77] A good example for this problem is that in the case that a certain model is designed with one specific tool which is not capable of reviewing these models at the same time it is necessary to export and import the model into another tool. [77] This step can be hard to achieve because most tools are not compatible with each other because they use a completely different format for import and export. The consequence of this problem is that before a review can start there needs to be a lot of time and resources put into the conversion of the model from the export format of the design tool to the import format of the review tool. [77] For some tools, this conversion is not possible at all and often it is therefore only possible to review an image of the artifact. Many review tools also do not provide a component to highlight, include, exclude or comment on different areas of the model to report potential defects, split up tasks, or mark already reviewed areas. [77]

In section 2.5 we already presented different tools which are capable of reviewing different artifacts. The main focus in the development of the MDRE was to implement an

ID	Feature MDRE	Initial state
General		
F1	Projects	✓
F2	User management	✓
F3	Audit logging	✓
Models		
F4	Model configurations	✓
F5	Model design	✓
Reviews		
F6	Review assignments	✓
F7	Reviews	✓
F8	Defect reporting	–
Experimentation		
F9	Experiments	–
F10	Normal tasks	–
F11	Review tasks	–
F12	Google Forms tasks	–
F13	Submissions	–
F14	Submission table	–
F15	Task overview	–

Table 5.2: Features of the MDRE in its initial state [56]

application that solves the mentioned problems, takes over the most important advantages of the presented tools, and combines the design and review process in a single tool. Furthermore, some disadvantages of presented review tools like the missing ability for reviewing artifacts that are not source code like models should be solved by the MDRE. During the software development process of the MDRE, the planned features were collected during different workshops with domain experts. Prock et. al. in [56] collected all required features and estimated the development effort of each feature of the MDRE. Table 5.2 shows an overview of all features that had been implemented in the initial state of the MDRE at the start of this work. Every individual feature available in the initial state is explained later within this section.

5.1.2 Architecture & Technologies

The MDRE prototype is a distributed software system that is split into a backend component, three different databases which store the backend data, and a frontend component. [77] In figure 5.1, all individual components and the communication paths of the MDRE are displayed.

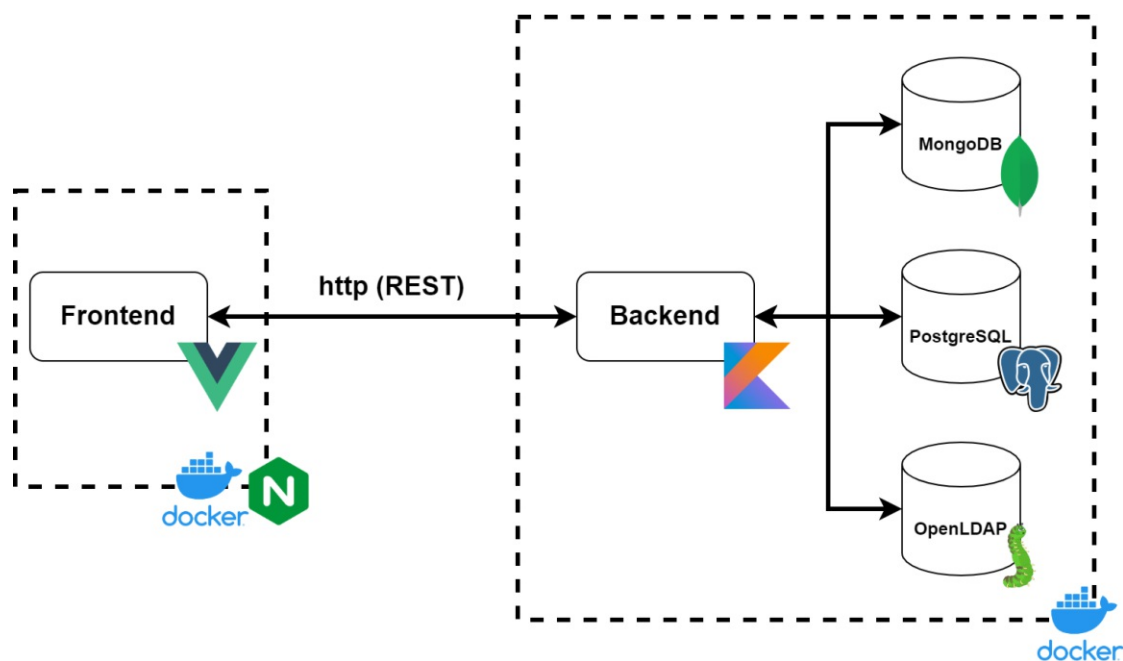


Figure 5.1: Architecture of the MDRE

Backend

The backend component of the MDRE is responsible for the communication with the frontend. A REST API is provided by the backend which the frontend connects to via the HTTP protocol. [77] With the help of the provided API, the backend sends and receives data from the clients of the MDRE users and processes the received data within its service layer. The whole communication between the backend and the frontend is completely stateless. [77]

For authentication, Java Web Tokens (JWT) [30] are used. A JWT stores all session information of a user which is in the case of the MDRE username, full name, all assigned roles of the user, and the expiration date of the session. Initially, the JWT is requested by the client to provide the username and the password of the respective user to the server. If the credentials are correct then the server sends a new JWT with all information of the logged-in user to the client. After the initial log in the JWT is added to each request within the HTTP header in the authorization field as a JWT Bearer token. [30] To make sure that the JWT cannot be manipulated and that the server can verify the correctness it is signed with a secret by the server. [30]

The backend stores the different data entities of the MDRE in three different databases. Data is read from or written to the correct database which is responsible for storing on a request of an MDRE client. For persisting the data the following databases are used in the MDRE:

- **PostgreSQL:** The PostgreSQL (PSQL) database is a powerful and open-source relational database management system (DBMS). [79] PSQL supports all functions and standards of the SQL standard. [79] Relational databases are one of the most popular DBMS in the world. [31] The big advantage of them is that data stored in a relational database can be put into relation to each other which provides an efficient and fast way of querying data. [31]

In the MDRE the relational database PSQL is used for storing projects, review assignments, reviews, audit logs, and so on. This kind of data can be stored efficiently in this type of database because there are many relations between these entities and the mentioned entities always follow the same scheme.

- **MongoDB:** MongoDB is a high-performance and open source NoSQL database focused on storing documents. [79] It is easy to use, easy to deploy, and especially suitable for storing large amounts of unstructured data. [79] The biggest advantage of MongoDB is the use of documents rather than rows which are stored in a binary JSON format and do not have to follow a fixed database scheme like in relational databases. [3] This type of data model is intuitive and more flexible because it does not require joins to collect data for one entity as everything regarding an individual entity is stored in the same document. [3]

In the backend of the MDRE MongoDB is mainly used to store entities that do not follow the same data scheme for every single instance stored in the database. Especially models and model configurations are highly customizable and therefore it would be hard to use a relational database for these types of entities. Another reason is that during the development the possible parameters of certain entities can change, which makes a document-oriented database a good solution because no scheme needs to be adapted on data structure changes.

- **OpenLDAP:** LDAP itself stands for Lightweight Directory Access Protocol and was originally designed as a network protocol but quickly became known as a whole directory architecture with its own specification. [12] The main focus of LDAP is to store different kinds of information about different kinds of entities. Even though LDAP is nowadays mostly used to store information about users it is not limited to that. [12] In LDAP, developers can clearly define which data should be stored in the directory. For example, a user could be defined with the first and the last name, a birth date, an address, a telephone number, a gender, and an email address. [12]

In the MDRE OpenLDAP, which is one of the fastest open source LDAP directory servers in the market, is used for storing user information including their passwords in a central directory. [12] The main advantage of using LDAP for authentication in the MDRE is that many authentication libraries and frameworks support LDAP out of the box. This means integrating OpenLDAP into the authentication procedure can be as easy as connecting to an OpenLDAP server via the LDAP protocol. The

bride support in the integration makes the professional user management of the MDRE easy to accomplish.

The backend of the MDRE has been developed in the programming language Kotlin. Kotlin is a modern programming language fully compatible with Java with various design improvements that provide advantages over Java. [77] It is open-source and available under the Apache 2.0 license. [28] The main goal of Kotlin is to provide a more productive and safer alternative to Java. An important advantage of Kotlin compared to Java is the provided safety. Certain errors which are hard to completely avoid by hand in Java are avoided by Kotlin automatically. [28] One example for such an error would be a `NullPointerException` which happens if a null object is accessed during runtime. This could lead to the crash of the whole application in the worse case or to an unknown error whose origin is often hard to spot. [28] Kotlin also provides various syntactical improvements like the possibility to combine object-oriented and functional programming code easily. [28]

The mentioned advantages over Java and further improvements led to the decision of using Kotlin as the main programming language for the MDRE backend. To save resources in the development process the Spring Boot framework is used. [77] Spring Boot is an enterprise framework that provides a lot of functionality to support the developers. The main focus of the Spring Boot framework is the separation of concerns in which the framework takes over a lot of work from the developers. [77] For example, it comes with implementations that provide a full REST API implementation, which just needs to be called and customized by the developers saving the resources of implementing everything on their own. [77]

For the build and package process of the MDRE backend, the software project management and comprehension tool Apache Maven is used. [77] Apache Maven builds and packages applications according to the customization of the developers done in the Maven configuration file. Another important advantage of Apache Maven is dependency management which also is used in the MDRE backend. With the help of the Maven dependency management, all dependencies like Kotlin and the Spring Boot Framework are imported automatically. [77] There are no extra coding tasks involved, Maven works based on its configuration files. [77]

Frontend

The MDRE frontend is a dedicated component that communicates with the MDRE backend using the already presented REST API via the HTTP protocol. All information that is displayed within the frontend is loaded from the backend at runtime.

The frontend is implemented as a single-page web application that is served via a web server but executed only on the client-side to save server resources. [77] The reason for choosing a framework that supports the creation of single-page web applications is that these are becoming more popular in the last few years. [77] Well-known frameworks

behind these single-page web applications are for example Angular, React, and VueJS. These frameworks are known to be easy to use and assist in the creation of professional web applications because they take over work from the developers. In particular, the framework chosen for the implementation of the MDRE frontend is VueJS. [77] The main reason for the usage of VueJS is that there are various SVG libraries available that facilitate the implementation of the editing and drawing of models. [77]

VueJS itself is only a web framework that is installed via the NPM package manager and there exists only a development web server but no production web server. [77] Therefore, the webserver Nginx is used to distribute the compiled frontend web applications to the clients the execution itself happens on the client-side. [77] Nginx is a well-known and open-source web server that fulfills modern security standards and can be used in a production environment.

Containerization

The MDRE uses many components with different technologies. Therefore, if the installation of all MDRE components on a server would be done by hand it would be a laborious task with many steps and likely an extensive manual would be needed to be able to deploy the MDRE on a server. To simplify this deployment process the MDRE uses the tool, Docker. Docker is a container virtualization tool that is lightweight and does not use as many resources as traditional virtual machines with their own operating systems installed. [1]

During the deployment of the MDRE backend, the three databases which have already been explained previously are deployed into their own containers. There is one container for every individual database instance. The data which is stored within the containers is persisted with the help of volumes and stored on the host operating system. [77] For the deployment MDRE backend itself, a maven docker container is created which builds and packages the backend application and copies over the compiled executable into a separate Java runtime environment container to save space and resources. [77] In this container, the backend application can eventually be executed. [77]

Similar to the backend also the frontend is built and deployed using Docker. First, the frontend is built within a Node.js container. [77] Once the build is completed the compiled files are copied to another container that runs an Nginx image. From this Nginx container, the clients can access the frontend and execute it. [77]

With the help of docker, the deployment of all components of the MDRE can be reduced to a few commands that need to be entered into the command line of the destination server and only takes less than an hour to finish.

5.1.3 Features & Functionality

In this section, we present the features and functionalities of the MDRE at the time of the start of this work. Figure 5.2 displays a simplified class diagram with the most important entities of the MDRE and their relations to each other. This diagram gives an overview of the features and functionalities which have been implemented into the MDRE.

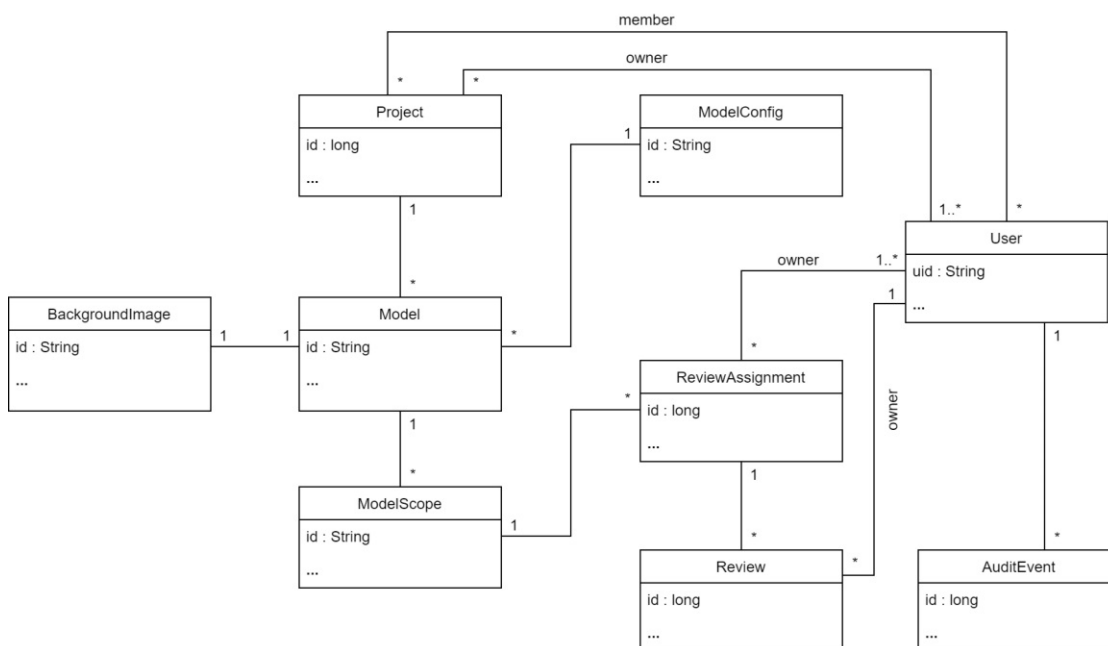


Figure 5.2: Simplified class diagram of the initial MDRE

In the following sections, we present every feature (see table 5.2) in detail and explain what the feature is used for. The initial state of the MDRE includes the features F1-F7.

General Features / Characteristics

In this section we present the MDRE features of the general category (see table 5.2).

Project (F1): After the successful login into the MDRE, the project overview as displayed in figure 5.3 is visible. A project is created by users and offers access restrictions to certain entities in the first place. In a MDRE project, there needs to be at least one project owner and there also can be multiple project members. While owners can change every setting and invite more participants to the project the abilities of project members are limited for security reasons. In the MDRE projects are the central concept for working with models and reviews. Within a project, a user can design different models based on individual model configurations and open review assignments for certain parts of the designed models. These review assignments later need to be executed by participants

within the project by creating reviews according to the underlying assignment. Neither models, reviews nor review assignments can exist without an underlying project, therefore projects are an important concept in the MDRE.

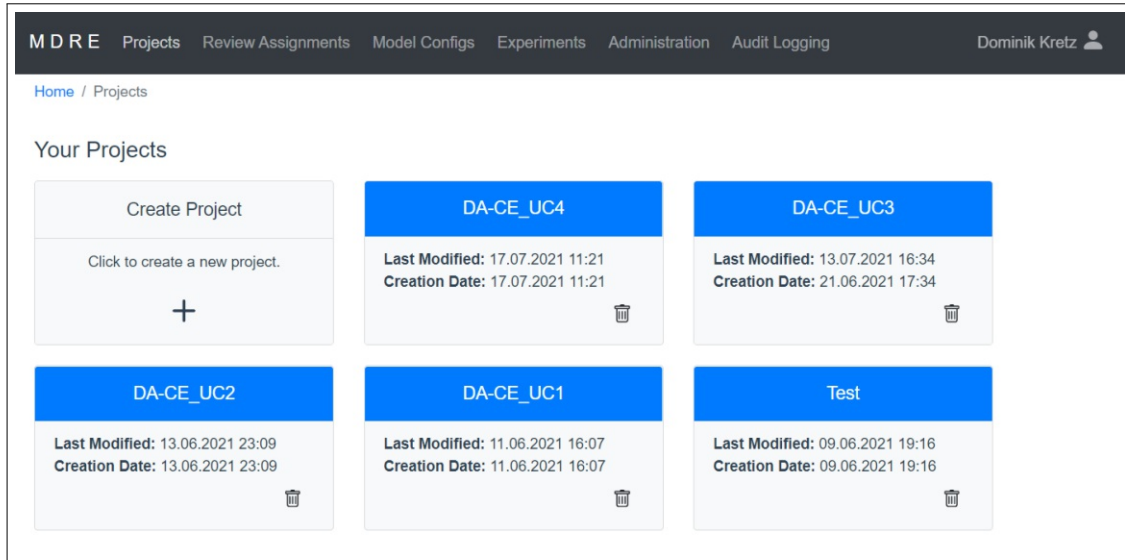


Figure 5.3: Overview of projects

User management (F2) & Audit Logging (F3): The MDRE offers the possibility of a full user management that gives administrators the possibility to add, change and delete user accounts. Within the user management, individual users can be assigned to different groups that provide privileges for certain features of the MDRE.

An important security feature of the MDRE is audit logging that is performed based on the user management. During the logging, data and information is collected about certain events which happen on the MDRE server. [67] The logs should give information about who has logged into the MDRE and which actions have been taken by the individual user. [67] Since the audit logging is done on the server-side there is no way for the users to tamper or deny the collection of the actions which have been carried out by them.

Audit logs can help to investigate certain incidences in case something has gone wrong or a user account has been compromised. In figure 5.4, the audit logging overview of the MDRE is displayed. By default, the first actions logged are displayed and there also exists the possibility for filtering out certain time frames which are especially of interest.

```

[{"id":1,"userId":"dominik","entityId":null,"entityName":"UserTag","action":"READ","reviewState":null,"created":"2021-06-08T12:59:13.265"},
{"id":2,"userId":"dominik","entityId":null,"entityName":"Tag","action":"READ","reviewState":null,"created":"2021-06-08T12:59:13.45"},
{"id":3,"userId":"dominik","entityId":null,"entityName":"ReviewAssignment","action":"READ","reviewState":null,"created":"2021-06-08T12:59:14.437"},
{"id":4,"userId":"dominik","entityId":"2","entityName":"Project","action":"CREATE","reviewState":null,"created":"2021-06-08T13:00:40.464"},
{"id":5,"userId":"dominik","entityId":"2","entityName":"Project","action":"READ","reviewState":null,"created":"2021-06-08T13:00:41.31"},
{"id":6,"userId":"dominik","entityId":null,"entityName":"ReviewAssignment","action":"READ","reviewState":null,"created":"2021-06-08T13:00:41.436"},
{"id":7,"userId":"dominik","entityId":"2","entityName":"Model","action":"READ","reviewState":null,"created":"2021-06-08T13:00:41.455"},
{"id":8,"userId":"dominik","entityId":"2","entityName":"Model","action":"READ","reviewState":null,"created":"2021-06-08T13:00:41.505"},
{"id":9,"userId":"dominik","entityId":null,"entityName":"ModelConfig","action":"READ","reviewState":null,"created":"2021-06-08T13:00:41.528"},

```

Figure 5.4: Audit Logging

Model Design Features / Characteristics

In this section we present the MDRE features of the models category (see table 5.2).

Model Configuration (F4): Model configurations represent the basis of model design. A model configuration is a JSON file of a special format in which all details for the model design need to be specified. In such a configuration, nodes and edges must be defined by configuring their appearance as SVG code, parameters of the entities need to be specified and rules for the model design have to be added. An example of a common rule for model drawing would be the specification of which edge type is allowed to connect nodes of one type to nodes of the same or another type. In figure 5.5, the overview of a model configuration in the MDRE is displayed showing all node and edge types including their attributes defined within this configuration.

Model Design (F5): Designing models is one of the main features of the MDRE. As already mentioned models do not use any kind of hard-coded designs like predefined node or edge types, they are completely customizable using model configurations written in JSON.

In figure 5.6, the model design editor in which models can be drawn is displayed. On the left side of the figure the different types of nodes which can be used within this model are shown, as well as the most important settings of the editor. This left settings pane acts as the main menu bar of the model review editor. A new node of a certain type can be created by dragging and dropping it from the menu bar into the editor pane. In the center of the figure the already designed model is displayed showing both the nodes and the edges within it.

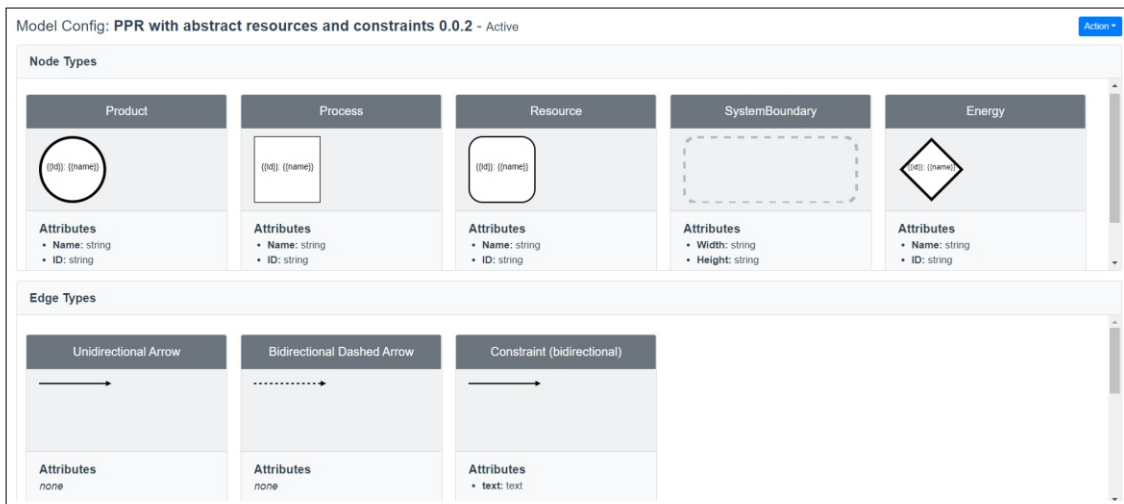


Figure 5.5: List of model configurations

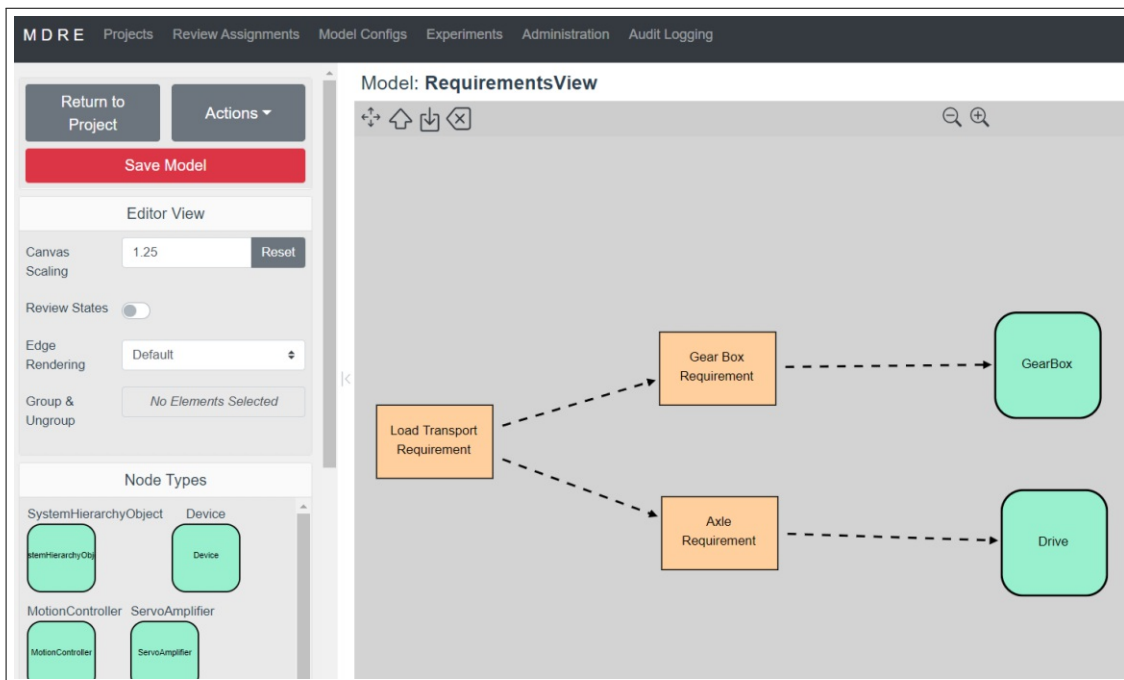


Figure 5.6: Model design view

The main problem in designing models is that there are many different specifications of models, for example, the UML model specification alone has a big variety of model types. Therefore, it is not possible to support the design of every model specification that exists in the whole world with the generic concept of the MDRE model configurations. However,

to overcome this problem in the MDRE it is possible to import models which have been drawn in a third-party tool as images. This makes it possible to at least partly support every possible model available which can be displayed as an image which is especially important when the specific model should be reviewed.

Model Reviewing Features / Characteristics

In this section we present the MDRE features of the reviews category (see table 5.2).

Review Assignment (F6): To review an area in a model within the MDRE review assignments are required. A review assignment is the basis for every review which is opened. Therefore, a review assignment has to specify all details about a planned review. The most important parameters which need to be defined in a review assignment are the area of the model that should be reviewed, the name of the assignment, the task description, and the method instructions which reviewers have to follow. Furthermore, also additional materials can be provided which are needed for conducting the review like the specifications of the model and some tags can be added which are for example skills or qualifications reviewers must fulfill. One example for such a tag would be that if we review a model which is designed in Spanish every reviewer must have the tag Spanish to be able to start a review based on the assignment.

Another important feature of the MDRE review component is the possibility of performing a majority voting, therefore in a review assignment, a target review count needs to be set. If for example a target count of three is specified only three different reviewers can start a review based on this review assignment. At a target count of three, the review assignment can either be *Approved* or *Rejected* based on what the majority of the reviewers state. Although, the state of a review assignment can be overwritten by the owner of the assignment which for example is also needed when there is no majority for either of the two results. In figure 5.7, the review assignment overview within the project details showing review assignments with all three different states possible is displayed.

Review Assignments		
Test	School	Class
Current State: In Process Creation Date: 18.07.2021 12:58	Current State: Rejected Creation Date: 18.07.2021 13:03	Current State: Approved Creation Date: 18.07.2021 13:04

Figure 5.7: List of review assignments

Review (F7): In the case that a review assignment is in the state *In Process* and the reviewer has not opened a review for this assignment yet it is possible for him to open a new review. Figure 5.8 displays the view for conducting reviews that are opened after a new review is created. The screen during the process of performing a review is split into three different sections. In the left section (1) all details of the review assignment are displayed. This information provides instructions about how the review should be conducted and which tasks should be fulfilled during the whole process.

The main part in the center of the screen (2) shows the model under inspection. Within this model view the area of the model which should be reviewed is marked in yellow, this area is defined during the review assignment creation. The model view is used to inspect the area of the model under review, it additionally provides the possibility to mark certain parts of the model with different markers. Markers have the main functionality of providing the reviewer the possibility to mark his progress to keep the overview. There are four different markers available checkmark (Ok), crossmark (Defect), question mark (Unclear), flag (Possible improvement). In the right pane of the screen (3) the reviewers can leave a comment and close the review with a specific state. Defects can only be reported by describing them in detail using the comment functionality.

Reviews can either be closed as *Approved*, *Rejected*, or *Unclear*. In case there is no defect found the reviewer would use the *Approved* state and the *Rejected* in case there are important defects that need to be fixed. If the review or the model area under review is not clear to the reviewer then the *Unclear* state should be used to close the review. All results of the reviews count to the majority voting of the underlying review assignment as already explained previously. However, the *Unclear* state is of course neither positive nor negative and therefore considered as a neutral result.

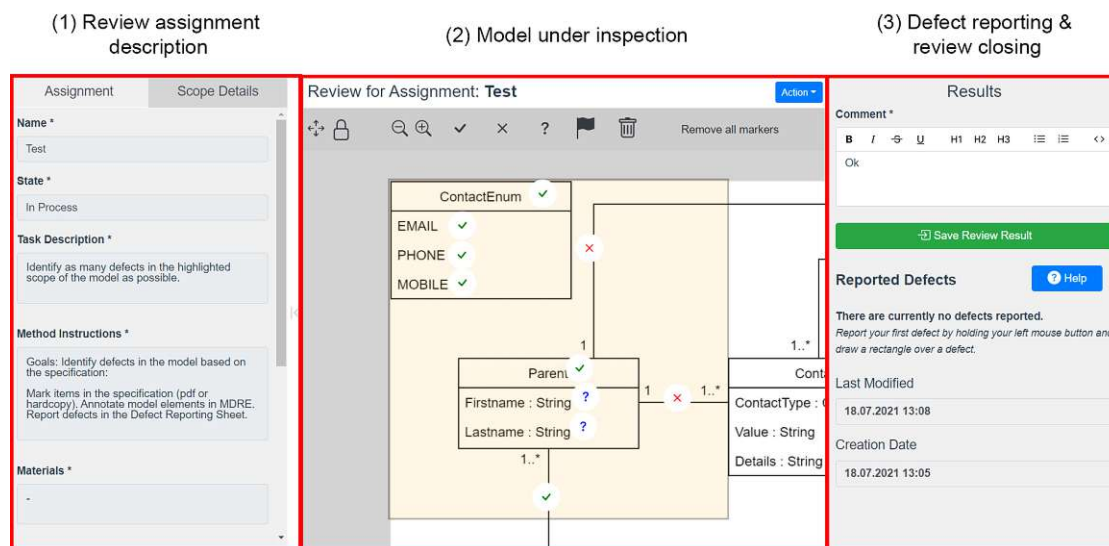


Figure 5.8: Model review view [77]

5.2 Tool Support AddOn for Model Reviewing (RQ1)

In this section, we present the tool support AddOn for model reviewing within the MDRE that has been implemented.

5.2.1 Introduction

Within section 5.1.3 we already explained the functionality of the model review tool support in the MDRE that was already available at the start of this work. The functionality of the state of the MDRE at the beginning did not provide any possibility for marking and reporting defects within models. It was only possible to mention and describe found defects in the comments and set the result of the review to *Rejected* to request a fix. However, even if the description of the defect within the comment section was detailed the owner of the review assignment could still overlook the report or fail to find the defect within the model in case of a bigger model.

The main requirement that was formulated in the RQ1 was to implement tool support that offers the possibility to provide detailed model defect reports to the review assignment owners within the MDRE. Therefore, we advanced the already existing tool support for model reviews and implemented a possibility for the reviewers to highlight and describe defects within the reviewed models by simply marking them within the model view. Once a defect has been reported it is persisted on the server within the backend and can be accessed by all review assignment owners. The owners can after the review has been finished browse through all reported defects and check each defect individually. In case a reported defect is not a false-positive and therefore a real defect then the review assignment owners can analyze this defect and decide if it needs to be fixed and within which time frame a fix needs to be applied depending on the severity of the defect.

In the following section, we present all features of the MDRE model review AddOn component. Table 5.3 shows a comparison between the features that have been available in the initial state of the MDRE and all features that have been added with the development of this additional component for model review tool support.

5.2.2 Features & Functionality

In this section, we present the features and functionalities which are offered by the AddOn component for model review tool support. Figure 5.10 displays a simplified class diagram with the most important entities of the MDRE and their relations to each other after the AddOn for model review tool support was implemented. The diagram shows all entities which had already been implemented into the MDRE before the start of this work with the color white. Additionally, all entities that belong to the AddOn for model review tool support and that have been implemented are displayed with the color green. The model review AddOn component of the MDRE includes the feature F8.

Figure 5.9 shows a flow chart diagram that presents the process of creating, executing, and administrating reviews within the MDRE. A more detailed version of this flow

ID	Feature	Initial state	Review AddOn
General			
F1	Projects	✓	✓
F2	User management	✓	✓
F3	Audit logging	✓	✓
Models			
F4	Model configurations	✓	✓
F5	Model design	✓	✓
Reviews			
F6	Review assignments	✓	✓
F7	Reviews	✓	✓
F8	Defect reporting	–	✓
Experimentation			
F9	Experiments	–	–
F10	Normal tasks	–	–
F11	Review tasks	–	–
F12	Google Forms tasks	–	–
F13	Submissions	–	–
F14	Submission table	–	–
F15	Task overview	–	–

Table 5.3: Feature level comparison of the MDRE (after model review AddOn). All features are displayed in figure 5.10. Newly added features are displayed in green.

chart can be found in appendix A of this work. Within the flow chart we distinguish between the two roles of review owners and reviewers. Review owners start with stage 1 by choosing an area (scope) within a model that should be reviewed. After that in stage 2, they set up a review assignment for the selected scope and enter the full review description and the review guidelines for the reviewers. Once all reviewers closed their review at the end the review owners are responsible for evaluating all reported defects within stage 3.

The reviewers execute the review assignment description and report possible defects within the selected model scope. Therefore, each reviewer opens a review at the beginning within stage 4. Once the review is opened the reviewers follow the review assignment instructions in stage 5 and report possible defects to the review owners in stage 6. These two stages are repeated until every part of the model scope is checked. In the end, the reviewers close their reviews within stage 7.

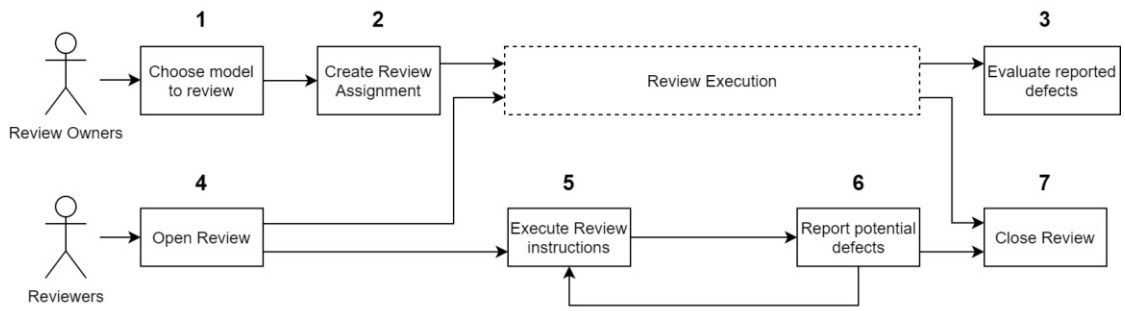


Figure 5.9: MDRE Review execution flow chart

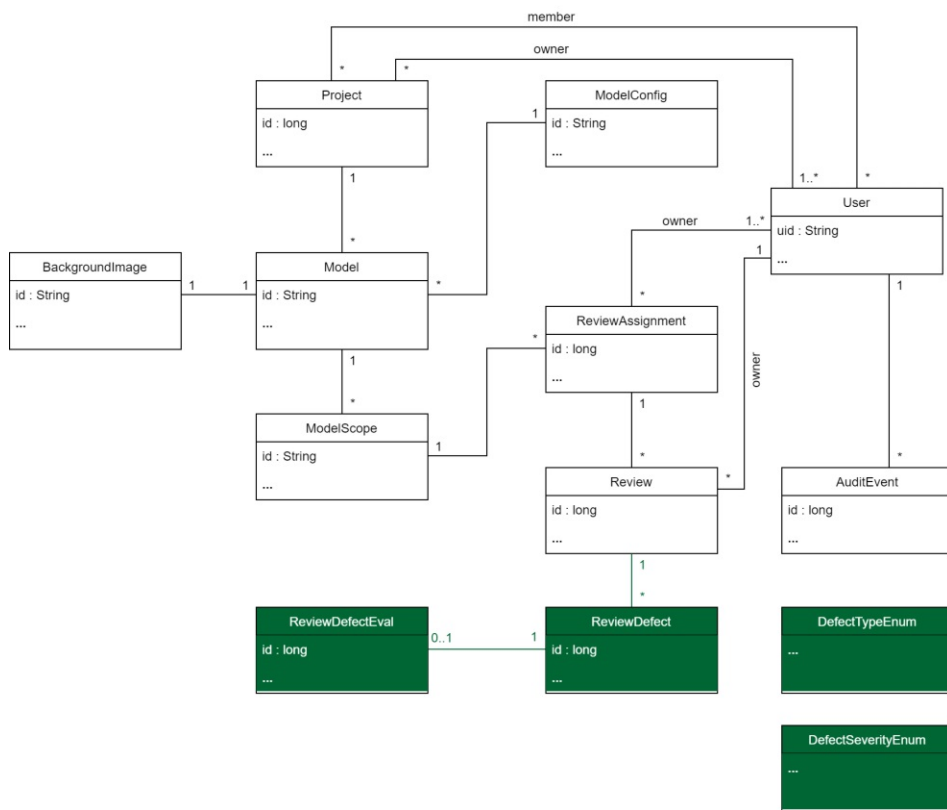


Figure 5.10: Simplified class diagram of the MDRE with the AddOn for tool-supported model reviews (green)

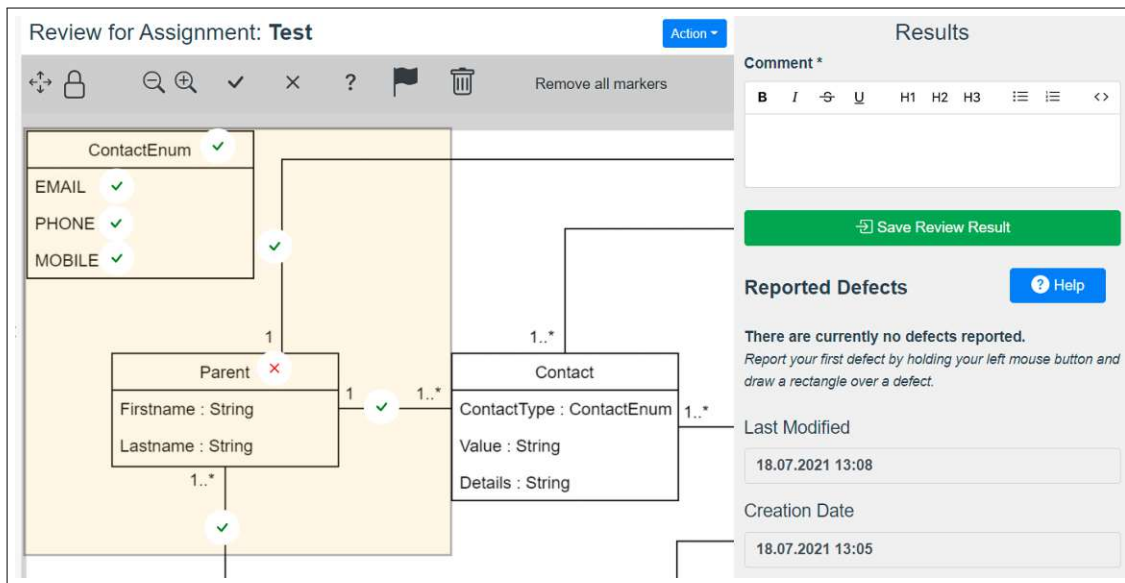


Figure 5.11: Review view of the MDRE (Stage 5 in flow chart 5.9)

Model Reviewing Features / Characteristics

In this section we present the feature of the model review AddOn component of the reviews category (see table 5.3).

Defect reporting (F8):

In figure 5.11, we see the already presented review view of the MDRE with a model in which parents of students are mapped to a certain amount of contacts that can be used to get in touch with the parents. Additionally, an enum for contact types is defined which states the type of the contact that is stored for every individual contact of a parent. It can be observed that the enum and the parent entity are highlighted in yellow which means that both highlighted entities should be inspected within this review.

The reviewer working on the review displayed in figure 5.11 already placed some markers to label which parts of the model area should be inspected has already been checked by him. However, as we already mentioned in the previous section these markers have only the task to assist the reviewers during their inspections. None of the used markers are persisted within the backend on the server and therefore are lost in case the browser cache would be emptied on the client.

It is of obvious importance that defects are persisted within the backend and that details can be added to every individual defect. Within the example model which we already presented there has been an error added into the parent entity. Within this entity, the gender attribute which should define the gender of each parent is missing. Therefore, this is a (potential) defect and needs to be reported using the implemented model review tool support AddOn. In figure 5.12, the process to report a defect is displayed. To report a

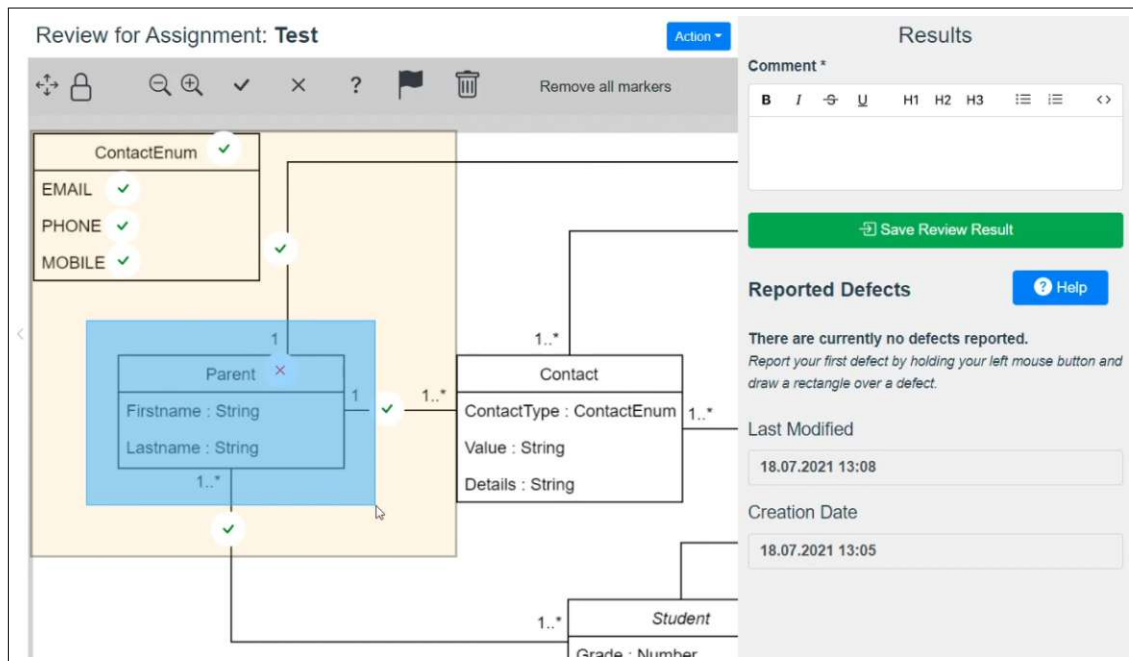


Figure 5.12: Highlighting a defect within the review view of the MDRE (Stage 6 in flow chart 5.9)

defect within the review view the inspector can simply hold down the left mouse button and draw a rectangle selection box around the area that contains a defect. The reviewer must highlight a potential defect as precisely as possible to make sure that the review assignment owners can later locate the exact position of the defect without any additional effort.

Once the rectangle selection box is drawn by releasing the left mouse button the defect report form is automatically opened. Within this form, the reviewer can describe all details of the defect and send the report to the backend. In figure 5.13, the defect reporting form of the MDRE is shown. Within this form there are the following individual details that a defect report needs to contain:

- **Defect description:** This field should contain a short but detailed description of the defect. The description mentions in which form and why the selected area marks a defect including all details to its appearance. The only way to communicate the background of the defect to the review assignment owners is by providing a good and accurate defect description.
- **Position in specification:** Usually, models are based on some kind of specification like a PDF document that specifies the exact scenario and details that the model should map. The position of a defect within the specification is important because it helps the review assignment owners to understand why the reported area could

be a defect. Therefore, in this field an accurate line number in the document referencing the location where the specification is different from the model.

- **Defect type:** The type of the defect describes what kind of defect we are dealing with. There currently can be four different types of defects selected in the MDRE defect reporting form:
 - *Missing:* The reported area does not contain details that are described in the specification.
 - *Wrong:* In the selected area there is a component that does not match with its description in the specification documents.
 - *Unclear:* The reviewer does not understand the selected area in the model and therefore cannot check if it matches the specification. This type does not need to be a real defect but can point to areas of the model which need to be improved and made more clear.
 - *Superfluous:* Within the reported area of the model there are details displayed that are not part of the specifications documents and therefore are superfluous and can likely be removed by the review assignment owners.
- **Defect severity:** The defect severity specifies the level of the impact a defect can have on the whole artifact according to the reviewer and therefore how important it needs to be fixed. With higher importance, it is necessary to fix a defect faster and invest more resources into the fix than with lower severity defects. In the MDRE defect reporting form there are currently three different severity categories that can be chosen for a defect that should be reported:
 - *Minor:* A minor defect that does not have much impact on the design and operation of the artifact. Therefore it does not require urgent fixing or a lot of resources that need to be put into fixing this defect.
 - *Important:* The severity category important marks defects which should be fixed quite soon and the fix itself should also be focused to avoid new defects resulting out of the fix.
 - *Critical:* Critical defects must be fixed as fast as possible as they can have severe impacts on the whole artifact which is under review. The fix should be carried out immediately and needs a maximum of attention to make sure that the defect is removed and there are no new problems that were invented as a result of the applied fix.

Once the defect is reported by clicking on the Save button all details including the position of the defect are sent to the backend. In figure 5.14, the filled-out defect reporting form for the example defect which has been explained previously is displayed. Because of the missing data in the model we have set the defect severity of this defect to *Important*.

The screenshot shows a 'New Defect' dialog box with the following fields:

- Defect Description ***: A text input field containing the placeholder text 'Defect Description'.
- Position in Specification (Line number in PDF) ***: A text input field containing the placeholder text 'Position of defect'.
- Defect Type ***: A dropdown menu with the placeholder text 'Choose a defect type'.
- Defect Severity ***: A dropdown menu with the placeholder text 'Choose a defect severity'.

At the bottom right, there are two buttons: 'Cancel' and 'Save'.

Figure 5.13: Defect reporting form (Stage 6 in flow chart 5.9)

The screenshot shows the 'New Defect' dialog box with the following example input:

- Defect Description ***: 'Gender is missing in Parent entity'.
- Position in Specification (Line number in PDF) ***: '4'.
- Defect Type ***: 'Missing'.
- Defect Severity ***: 'Important'.

At the bottom right, there are two buttons: 'Cancel' and 'Save'.

Figure 5.14: Defect reporting form with example input (Stage 6 in flow chart 5.9)

This should make sure that the review assignment owners who are responsible for the review focus on a prioritized fix of this defect.

After a defect has been successfully reported to the backend it is displayed in the right sidebar of the review view within the MDRE. Figure 5.15 shows the previously reported defect that states the missing gender attribute of the parent entity that has been specified in line 4 of the specification for the example model. The two buttons on the right side of a defect provide support for editing the defect further or in case it was reported by accident also the deletion of a defect is possible. Within this sidebar, it is as mentioned also possible to leave a comment and to save the review result via the button just below the comment field.

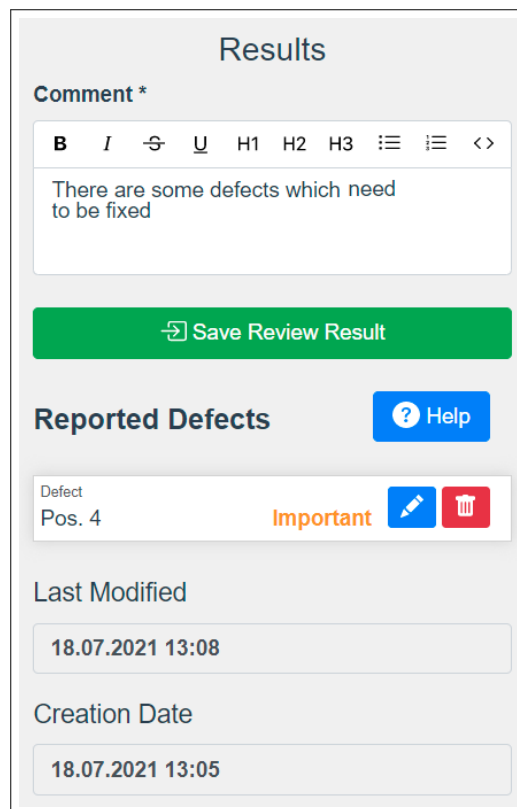


Figure 5.15: List of reported defects and review finalization (Stage 7 in flow chart 5.9)

Saving a review was already possible at the beginning of this work but along with the improvements for reporting defects in models during a review, we also improved the appearance and usability of the save review dialog that is used to close / complete reviews. Figure 5.16 shows the new and improved dialog to save reviews in the MDRE with the three possible outcomes a review can have after it was completed. It is also important that a review does not need to be closed instantly but also can be left open and be continued later. Previously, the feature of selecting one of the 3 different states or leaving

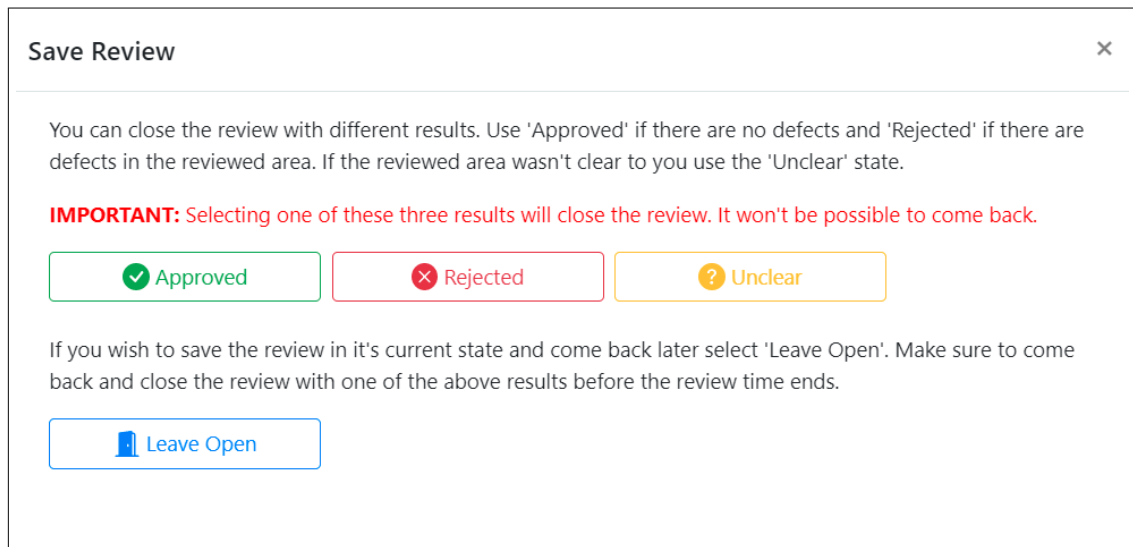


Figure 5.16: Dialog to save and close a review (Stage 7 in flow chart 5.9)

the review open was realized by one button and a drop-down selection with the four mentioned possibilities. This was described as confusing by some users and so the whole dialog needed an update. The new version of the dialog for this feature is an improvement and makes it easier to use by the inspector because now each outcome has its own button and therefore the selection of the correct state can no longer be missed out.

Once a review has been closed it is shown in the review assignment details and can also be accessed again by the owner of the review but no longer edited. Also, all owners of the review assignment can access closed reviews without the possibility of editing them, they also can see the reported defects of all reviewers and therefore can investigate them. To avoid that other reviewers are biased by already completed reviews they are not allowed to access any reviews not owned by them. In figure 5.17, our example review has been closed in the state rejected as the missing gender within the parent entity needs to be fixed before the model is used further in the development of an artifact according to the MDD approach which was already presented in section 2.2.4.

As it was already presented at the beginning of this section in figure 5.10 we not just implemented the functionality of reporting review defects but we also created the possibility of evaluating the reported defects within the backend of the MDRE. Currently, this evaluation which offers the possibility for the review assignment owner to comment and link reported defects to a defect list is only available to the backend.

At the moment, no user interface component makes it possible to evaluate reported defects in the web browser available in the MDRE frontend. The reason for this is that the evaluation of defects within the MDRE was no requirement that was formulated. However, the possibility has now been implemented within the backend which makes sure that in case that this requirement exists in the future it can easily be introduced in

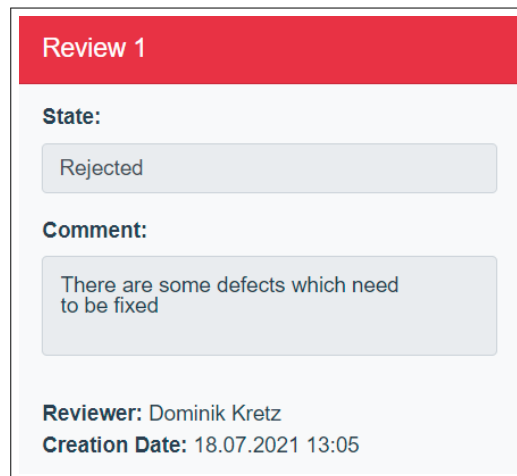


Figure 5.17: Rejected review in the review assignment overview

the MDRE frontend during a possible future work.

5.3 Tool Support AddOn for Experimentation (RQ2)

In this section, we present the tool support AddOn for experimentation within the MDRE that has been implemented.

5.3.1 Introduction

We presented in the previously displayed table 5.1, that there are many different software inspection tools with various features available. Each tool listed within this table is explained in detail within section 2.5. There is currently no software inspection tool available that supports both review and experiment administration as well as reviewing of models or images at the same time. Therefore, it is needed to use the MDRE as an application that provides tool support for review and experiment administration based on the existing features for model reviewing within it.

We explained the theoretical background of experimentation in section 3. Within subsection 3.3 we presented that every experiment consists out of five different phases which lead from phase E1 - scoping of the experiment to phase E5 - presentation and packaging of all data which has been collected during the experiment. Not each of the five phases can be supported with the help of a tool since tasks like the analysis and the interpretation of collected data are something that can hardly be automated with an application like the MDRE. However, all phases which require a lot of administration offer the possibility to automate the majority of all tasks within these phases with the help of tool support. It would for example be relatively easy to automate the allocation of different tasks for individual participants and to provide materials that are needed by all experiment participants to execute their tasks within the experiment.

The main requirement that was formulated in the RQ2 was to implement tool support that offers the possibility to support the administration and execution of reviews and experimentation within the MDRE. Within section 5.1.3 we already presented the functionality of the MDRE that was already available at the start of this work in detail. The functionality of the state of the MDRE at the beginning did not provide any possibilities for supporting and executing experimentation. Therefore, we analyzed every phase within experimentation and searched for tasks within all experiment phases that can be automated with the help of such an AddOn that provides tool support for experimentation in the MDRE. Along with the analysis different features that this tool support component should be capable of were formulated and collected for the implementation into the MDRE.

The main focus of the planned features for the experiment administration component supports the phases E2 experiment planning and E3 experiment operation. Both phases include a lot of administrative tasks as we already explained based on our use case diagram in section 1.1. These administrative tasks need to either be executed by the experiment managers or by the participants themselves. Therefore, providing tool support for these administrative tasks is important and saves time and resources before, during, and after experiments and the saved resources can later be used in different studies instead. As we also explained in section 1.1, without tool support experiments are executed with materials, tasks, and information spread over various platforms. With the tool support as many resources as possible are available in a single platform which is in this case the MDRE and the experiment participants are guided through all tasks with the possibility to see their overall progress within the experiment live. Especially the active guidance of all experiment participants is important to support the success of the experiment by avoiding that participants forget to execute tasks which would result in missing data within the experiment results.

In the following section, we present all features of the MDRE review and experiment administration AddOn component. Table 5.4 shows a comparison between the features that have been available in the initial state of the MDRE and all features that have been added with the development of the two additional AddOn components. Figure 5.18 shows a flow chart diagram that presents the process of creating, executing, and administrating experiments within the MDRE. A more detailed version of this flow chart can be found in appendix B of this work. Within this flow chart we distinguish between two different roles the experiment managers and the experiment participants. The experiment managers create the experiment within stage 1 and add different tasks to the experiment within stage 2. There are multiple different tasks available that can be added to an experiment, e.g., questionnaires, review tasks, and so on. During the execution of the experiment the experiment managers monitor the state of every participant within stage 3. At the end of the experiment all collected results are evaluated during stage 4.

The experiment participants first open their task dashboard within stage 5 and execute every task displayed that has been added to their dashboard during stage 6. They repeat stage 5 and stage 6 until all tasks have been completed. Finally, they complete the

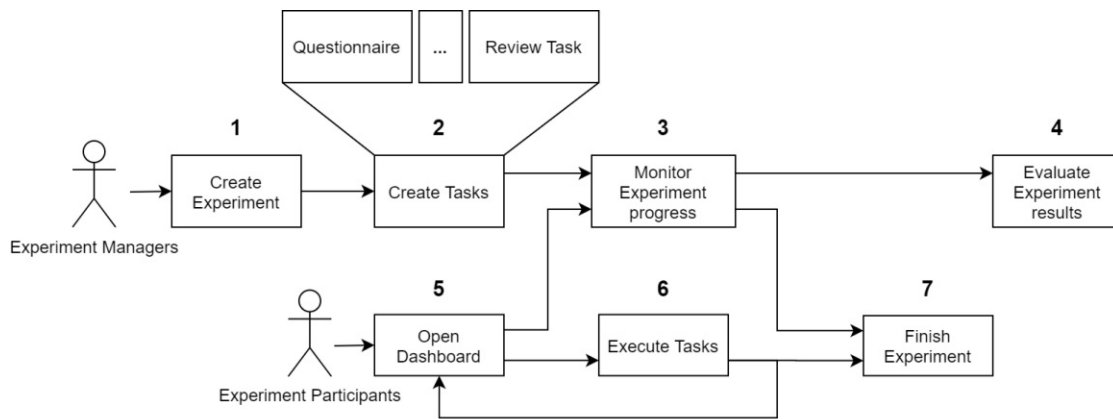


Figure 5.18: MDRE Review execution flow chart

experiment in stage 7 by handing in all materials.

5.3.2 Features & Functionality

In this section, we present the features and functionalities which are offered by the MDRE AddOn for experimentation tool support. Figure 5.19 displays a simplified class diagram with the most important entities of the MDRE and their relations to each other after the AddOn for model review tool support and the AddOn for review and experimentation administration tool support were implemented. The diagram shows all entities which had already been implemented into the MDRE before the start of this work with the color white. All entities that belong to the AddOn for model review tool support are displayed with the color grey. Furthermore, the entities that belong to the AddOn for review and experimentation administration tool support and that have been implemented are displayed in the color green. The MDRE component AddOn for experiment and review administration includes the features F9-F15.

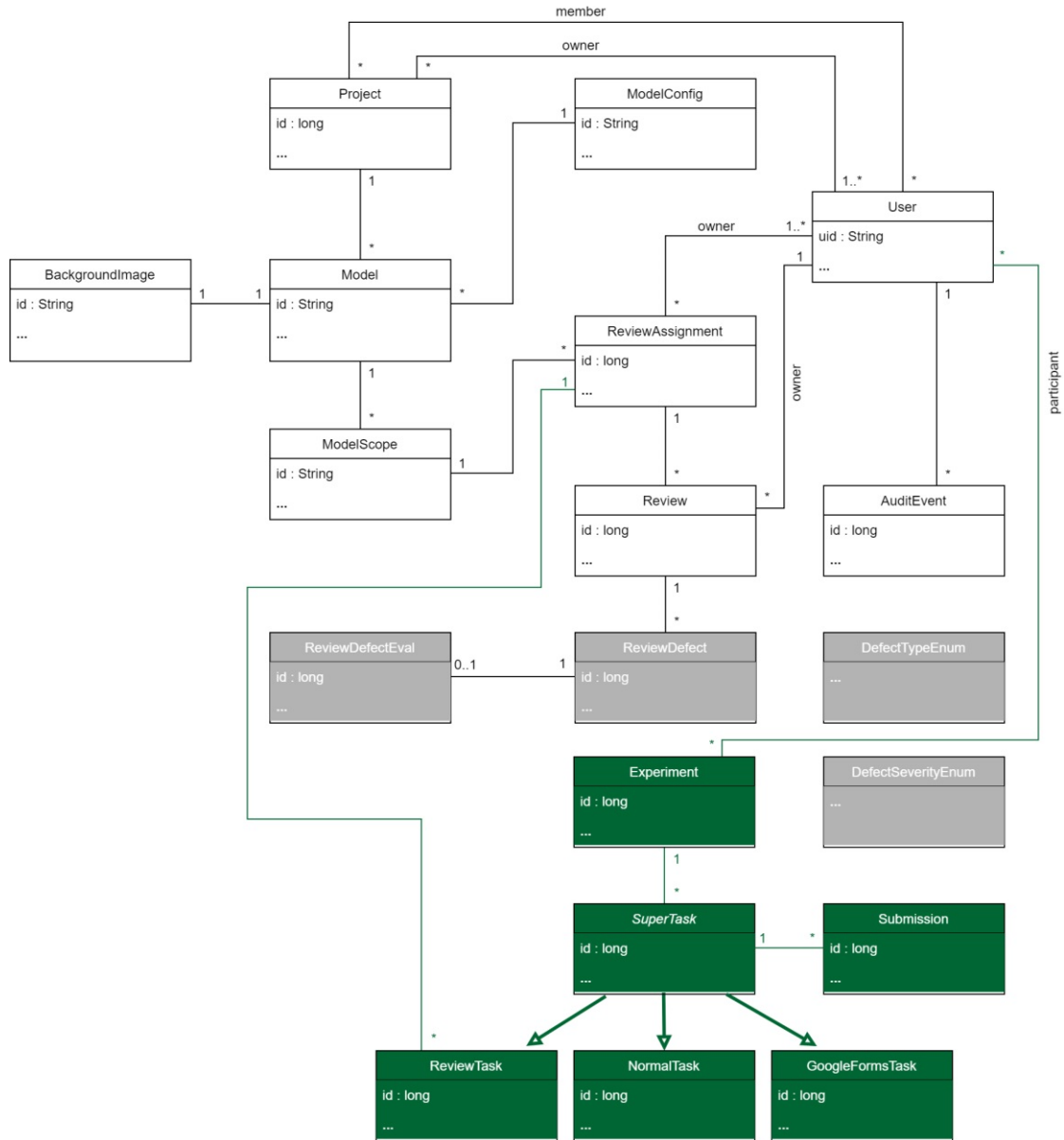


Figure 5.19: Simplified class diagram of the MDRE with the AddOn for tool-supported experimentation (green)

ID	Feature	Initial state	Review AddOn	Exp. AddOn
General				
F1	Projects	✓	✓	✓
F2	User management	✓	✓	✓
F3	Audit logging	✓	✓	✓
Models				
F4	Model configurations	✓	✓	✓
F5	Model design	✓	✓	✓
Reviews				
F6	Review assignments	✓	✓	✓
F7	Reviews	✓	✓	✓
F8	Defect reporting	–	✓	✓
Experimentation				
F9	Experiments	–	–	✓
F10	Normal tasks	–	–	✓
F11	Review tasks	–	–	✓
F12	Google Forms tasks	–	–	✓
F13	Submissions	–	–	✓
F14	Submission table	–	–	✓
F15	Task overview	–	–	✓

Table 5.4: Feature level comparison of the MDRE (after experimentation AddOn). All features are displayed in figure 5.19. Newly added features are displayed in green.

Experimentation Features / Characteristics

In this section we present the features of the experiment and review administration AddOn component within the experimentation category (see table 5.4).

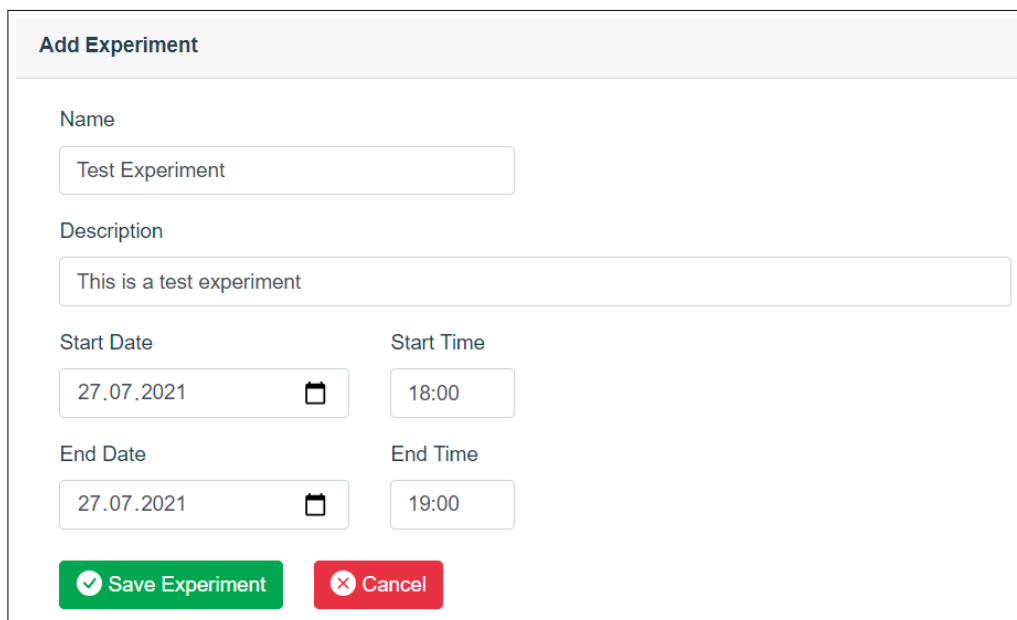
Experiments (F9)

In the tool support AddOn for experiment and review administration experiments are the main entities that contain all other for this AddOn relevant entities within them. This can also be seen in the already presented class diagram in figure 5.19. Experiments can be created by MDRE administrators and offer the possibility to assign tasks to individual experiment participants who are part of this experiment which takes place at a certain time. Every experiment stores the following types of information:

- **Experiment name:** The name of the experiment is the main identification for the experiment within the MDRE. It is displayed to both experiment participants and MDRE administrative users.
- **Description:** The experiment description can include additional information on the experiment for the participants. It is visible to the participants in case it is filled out but it is optional and does not have to be entered.

- **Start time:** After the start time the experiment is visible to the participants and the participants can start to work on the individual tasks within the experiment.
- **End time:** Between the start and end time experiments are visible and tasks can be worked on. However, after the end time, the experiment becomes invisible for the participants and all started tasks can be finished but no new tasks can be started anymore.
- **Experiment participants:** The experiment participants are MDRE users that have been assigned to the experiment. Every participant can view the experiments he is assigned to and can work on tasks that are part of these experiments.
- **Tasks:** Within an experiment tasks are stored which need to be worked on by every participant of the experiment. Tasks can have different types depending on the work which needs to be done to fulfill them. Currently, the MDRE offers the following types of tasks: *Normal tasks*, *review tasks*, and *Google Forms tasks*. The different task types which are available in the MDRE are described in detail later within this section.

Figure 5.20 shows the dialog that is used to add experiments to the MDRE. In our example, we added an experiment called "Test Experiment" which takes place on the 27th of July 2021 between 6 pm and 7 pm. Once the user clicks on the green save button the experiment is sent to the backend and is persisted there in case all validation checks are passed.



The screenshot shows a dialog box titled "Add Experiment". It contains the following fields and values:

- Name:** Test Experiment
- Description:** This is a test experiment
- Start Date:** 27.07.2021
- Start Time:** 18:00
- End Date:** 27.07.2021
- End Time:** 19:00

At the bottom of the dialog, there are two buttons: a green "Save Experiment" button with a checkmark icon, and a red "Cancel" button with an "X" icon.

Figure 5.20: Form to add an experiment (Stage 1 in flow chart 5.18)

Name	Start	End	Participants	Tasks	Actions
Test Experiment	27.7.2021 - 18:00:00	27.7.2021 - 19:00:00	3	3	Edit Delete

Figure 5.21: Overview of all experiments

Once the experiment has been persisted within the backend it can be accessed in the overview of the experiment. The experiment overview is displayed in figure 5.21 and already shows the previously added example experiment. In this overview there exists the possibility to add more experiments and also to edit or delete the already existing experiments. From this overview, it is also possible to navigate to the submission table that gives an overview of all experiments currently running in the MDRE including a list of submissions that have been handed in by the experiment participants of the individual experiment. The submission table is explained in detail later within this section.

If an experiment has already been added and is later edited it is also possible to assign participants to this experiment. Only the MDRE users with the administrator role and all participants can view and work on an experiment. For all standard MDRE users who have not been added as participants, it is not possible to view the experiment nor to work on the tasks assigned to it. In figure 5.22, the list of participants who have been added to the experiment is displayed. To support the experiment managers and to increase the usability of the MDRE participants can either be added to an experiment by selecting them out of a list of all MDRE users or it is also possible to upload a CSV file that contains all usernames that should be added as participants. This feature is especially important for bigger experiments with many different groups of participants that should be added to different experiments in the MDRE. In this case for every group, a CSV file can be prepared and uploaded to the MDRE, this makes sure that it is not possible to forget to add participants or to add them to the wrong experiment. Within the list of all participants there also exists an option to remove MDRE users from the list of the experiment participants.

Tasks

Every task is part of an experiment and specified work packages that have to be fulfilled by the participants. As already mentioned there are different types of tasks called *Normal tasks*, *review tasks*, and *Google Forms tasks*. Every of the listed types has special attributes and its own purpose but there are similarities in the attributes which are shared among all three task types. All properties that are similar overall task types are mapped by a superclass that is called *SuperTask* within the MDRE. The whole generalization mapping is presented in the class diagram that is displayed in figure 5.19. All three task types share the following attributes among each other:




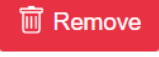
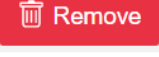
Participants				
				
Username	◆ Firstname	◆ Lastname	◆ Matriculation Nr.	◆ Actions
workshop01	workshop01	workshop01	01234567	
workshop02	workshop02	workshop02	01234568	
workshop03	workshop03	workshop03	01234569	

Figure 5.22: Experiment participants overview

- **Task name:** The task name becomes the main identification of the whole task. It is displayed to all experiment participants and the MDRE administrative users.
- **Submissions:** Every task stores multiple submissions of experiment participants. For every participant who starts or completes a task, an individual submission is stored. Submissions are explained in detail later within this section.
- **Short name:** Within the submission table that is explained later in this section a table showing all submissions of all tasks in an experiment is displayed. As this table would require too much space if the full name would be used every task has its own short name which consists out of a maximum of five characters. The short name is therefore only relevant for the MDRE administrative users and is never displayed to the experiment participants.
- **Description:** In the task description the participants receive exact details about what they have to do during this task. It should be as exact as possible and provide detailed information about the task, it is also possible to use URLs to external services for example for downloading materials within the description.
- **Maximum Points:** If a task is completed by a participant a submission is added to the task, these submissions that are explained later in this section can be graded. To be able to grade the task there must be a maximum amount of points specified that can be achieved within this task and may not be exceeded by an experiment manager who performs the grading.
- **Automated grading:** There is also the possibility of enabling automated grading. If this option is enabled within a task then every submission that is fully completed is automatically graded with the maximum points value. This feature is helpful for

example with simple tasks that do not require any point deductions like the upload of some materials which can either be done or not.

- **Task order:** The task order of a task is important for ordering the tasks on the dashboard for the participants. In the task dashboard that is explained later in this section, the user can see the upcoming tasks ordered by the task order in ascending direction. This value is helpful in case some tasks need to be fulfilled in a specific order.

In the next sections, we describe the individual task types and detail and what distinguishes them from each other.

Normal Tasks (F10): A normal task which is just called a task in the frontend of the MDRE is a task type that is not performing any kind of automated submission collection. Unlike the other tasks types, this type is not capable of automatically checking if an experiment participant has already started or completed a task. However, this task type is still important and useful because it can be used for many different kinds of tasks. An example of such a task would be the download of some kind of additional materials to support the experiment process from an external site as is displayed in figure 5.23. Another good reason to choose this type of task would be that the MDRE cannot support every third-party tool available because there are simply too many tools that could be relevant within an experiment. Therefore, a normal task tells the participants what they have to do within the third-party tool. The experiment managers then can manually check the third-party tool for the progress of the participants and add the submissions into the MDRE by hand. This does not work as efficiently as the other tasks type but it is a good solution in case there is no support in the MDRE available for the third party tool and this tool is needed necessarily.

This type of task has only one simple special attribute which is the possibility of setting the whole task to a placeholder. In case the placeholder option is enabled the task can no longer have any submissions. This is for example useful in case that the task only advises the participants to download some file as it is often not possible to check if an individual participant downloaded the material and also grading a download task does not seem to be necessary.

Review Tasks (F11): Within review tasks, the participants of an experiment are advised to execute an already in the MDRE available review assignment of a model. Therefore, the task is linked to the selected review assignment. The permissions to view the model and work on the review assignment are automatically granted for all experiment participants once the review task is created. Submissions for this task type are added automatically, for example when the review for the review assignment of this task is opened then a submission is created and is later updated when the review is set to completed by the participant. In case the automated grading option is enabled the maximum points for these tasks are automatically awarded once the review is completed, otherwise, the experiment managers can grade the submissions manually.

Figure 5.23: Form to add a normal task in an experiment (Stage 2 in flow chart 5.18)

Figure 5.24 shows an example of a review task with the two attributes that are special within this task type. The following two special attributes need to be set for this task type:

- **Project:** The project that contains the model and the review assignment that the review should be based on.
- **Review Assignment:** The review assignment for which a review should be created by the experiment participants during this experiment.

Google Forms Tasks (F12): In a Google Forms task, the participants of an experiment are advised to answer a questionnaire that is already available on Google Forms. Therefore, the task is linked to the Google Forms questionnaire. Questionnaires in experiments can be of importance for collecting the experience or feedback of participants which are later used to evaluate the results of the experiment. Submissions for this task type are added automatically when a participant has answered the questionnaire. This is done with the help of the MDRE backend which polls the Google Forms every few minutes and checks for new answers on the questionnaires. In case the automated grading option is enabled the maximum points for these tasks are automatically awarded once the questionnaire has been answered, otherwise, the experiment managers can grade the submissions manually.

Figure 5.25 shows an example of a Google Forms task with the three attributes that are special within this task type. The following three special attributes need to be set for this task type:

Task

Name: Short Name (max. 5 chars):

Description:

Select Project: Select Review Assignment:

Maximum Points: Order in Tasklist:

Automated Grading

Figure 5.24: Form to add a review task in an experiment (Stage 2 in flow chart 5.18)

- **Google Forms ID:** The Google Forms ID represents the unique ID of the whole form and is used to navigate the participants to the correct Google Form.
- **ID of the participant number field:** An important requirement to enable the mapping of MDRE users to Google Forms answers is the participant number of the experiment participants. All participants need to enter their participant number to make a mapping between their MDRE user and their answer possible. The ID of the participant number field is needed to find the participant number among all other fields within a single answer.
- **ID of the answer spreadsheet:** Google Forms itself does not provide any API to read answers from a form. However, Google Spreadsheet does offer the possibility of accessing spreadsheets via an API. Google Forms can be automatically synced to Google Spreadsheets which offers the possibility to access Google Forms answers via an API through Google Spreadsheets. Therefore, the public Google Spreadsheet ID is needed in the MDRE to define the location from which the backend can load the answers.

All tasks are listed within the experiment details. This list also provides the possibility of deleting and editing all tasks stored within the selected experiment. Figure 5.26 shows the example tasks that have been added previously in this section.

Task

Name Short Name (max. 5 chars)

Description

Google Forms ID ID of Matric. Number field ID of Answer Spreadsheet (PUBLIC)

Maximum Points Order in Tasklist

Automated Grading

Figure 5.25: Form to add a Google Forms task in an experiment (Stage 2 in flow chart 5.18)

Tasks					
Pos.	Name	Shortname	Points	Submissions	Actions
1	Experience Questionnaire	ExpQ	0	0	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
4	Download Materials	DM1	0	0	<input type="button" value="Edit"/> <input type="button" value="Delete"/>
5	Review the Test Model	RevM1	50	0	<input type="button" value="Edit"/> <input type="button" value="Delete"/>

Figure 5.26: Overview of all tasks in an experiment

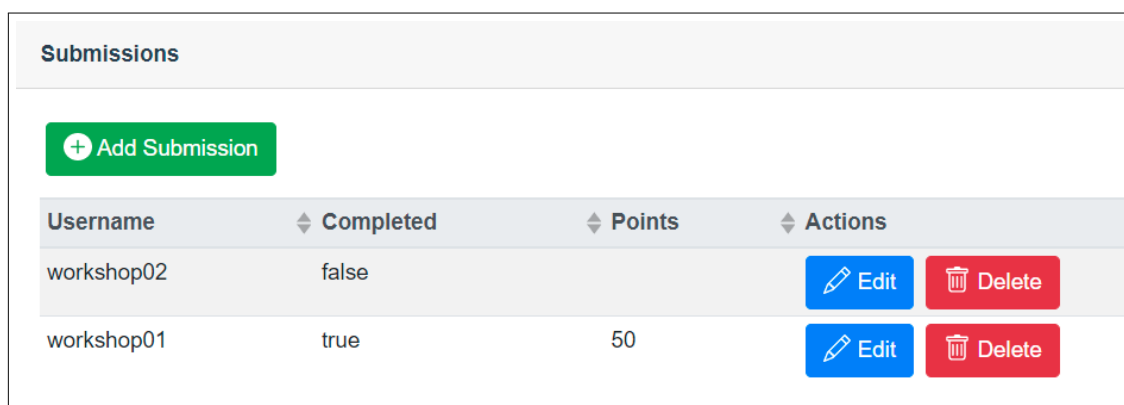
Figure 5.27: Adding / editing of a submission

Submissions (F13) Once a task is started or completed by an experiment participant a submission is added to the task. For every task, there can be one submission per experiment participant. Submissions are a quite simple concept and only have three different attributes which define them. The following three attributes are stored for every submission:

- **User:** The user field maps the submission to the MDRE user. Once a submission has been added the user mapping can no longer be changed afterward. An MDRE user can only be added to a submission in case that he is added as a participant in the experiment itself. In case that a user is not a participant of the experiment, it is also not possible to add any submissions to the task by this user.
- **Task completed:** A submission can be set to complete. Once a task is started a submission is automatically be added. If the submission is not set to complete it is considered as started. In case that the experiment participant has not started a task then there is also no submission available for the task at all.
- **Points:** Submissions provide the possibility to be graded optionally. The experiment managers can award points for certain submissions. However, in case that automated grading is enabled within the task, submissions that are in the state completed are automatically awarded the maximum amount of points defined in the task.

Figure 5.27 displays an example submission for the normal task that has been created previously. This submission has been done by the MDRE user called *workshop01* and is completed. Therefore, the submission was automatically awarded the maximum amount of points of the task which is 50 points.

All submissions of a task are displayed in a list within the task details. Figure 5.28 displays a list of all submissions that have been added to an example task previously. Submission can also be deleted or edited by experiment managers. The list shows two



Submissions			
+ Add Submission			
Username	Completed	Points	Actions
workshop02	false		Edit Delete
workshop01	true	50	Edit Delete

Figure 5.28: Overview of all submissions in a task

different submissions the first one has not been completed and therefore also has no points awarded. However, the second submission is the submission previously added in this section that is completed and has been awarded 50 points.

Submission Table (F14) For experiment managers, it is not efficient to check the progress of all participants during the experiment by browsing task by task in every experiment. This would take a lot of time and could hardly provide a good live overview of the progress of all participants. Therefore, an important requirement is the implementation of an overview that displays the progress of all participants live and without any additional actions needed to be performed by the experiment managers. In the MDRE this feature is called submission table and can be accessed via the experiment overview that was already presented in this section.

Figure 5.29 presents an example of the live view of an experiment within the submission table. For every task in an experiment, there is one column available showing the current state of all submissions. In general, all tasks are displayed within this table the only exceptions are the normal tasks that are set to placeholder because these types of tasks cannot have any submissions as already explained previously. Every user is represented in the table by an individual row. The three individual states of a submission are displayed by three different symbols. In case that the task has not been started by the user then this is indicated by a red cross mark if a task has been started two blue arrows are displayed and for completed submission, a green checkmark is shown. The number next to the symbols for submissions of review tasks represents the number of defects that the participant has already reported within the review. This offers the possibility of being able to view the live progress of each participant even during a review which usually takes more time than other task types. Without this feature there only would be the indication that a participant has started the review. As long as the review is not completed experiment managers would have no possibility of knowing if the participant is progressing on the review or is just stuck.

The submission table also offers the possibility to manually edit the submissions of

Experiment: Test Experiment							
User	Firstname	Lastname	ExpQ	RevM1	RevM2	FeedQ	
workshop01	workshop01	workshop01	×	×	×	×	×
workshop02	workshop02	workshop02	×	×	×	×	×
workshop03	workshop03	workshop03	×	×	×	×	×

Figure 5.29: Table of submissions within an experiment (Stage 3 in flow chart 5.18)

Change Submission
×

Change submission state:

✓ Solved

↻ Started

Close

Figure 5.30: Manual submission control within the submission table

participants for individual tasks. Therefore, it is only necessary to click on the symbol of the solution in the correct row and column according to the user and task for which the submission should be edited. After the click on one of the submission symbols that indicates the state of the submission for a user, a dialog is opened. This dialog is displayed in figure 5.30 and shows the buttons that trigger the state change. In this example there is no submission available for the selected task, therefore it is only possible to set the submission to solved or started via the buttons in the center of the dialog. This feature is important for experiment managers in case there are any problems with the automated submission check or if there are tasks that need to be checked by hand because they are not supported by the MDRE.

Figure 5.31 again presents the submission table of the previously introduced example experiment. However, this time the figure displays the submission table of the example experiment in a later state. Some of the participants already started and completed some of the tasks and there are already submissions available. For example, the MDRE user *workshop01* has already completed the experience Google Forms questionnaire (ExpQ) and the first review (RevM1) in which he reported five defects. He also has started the second review (RevM2) and already reported two defects within it. On the next line, the user *workshop02* has completed the first review with three reported defects and he

Experiment: Test Experiment							
User	Firstname	Lastname	ExpQ	RevM1	RevM2	FeedQ	
workshop01	workshop01	workshop01	✓	✓ 5	↻ 2	✗	
workshop02	workshop02	workshop02	✗	✓ 3	↻ 1	✗	
workshop03	workshop03	workshop03	✗	✗ 0	✗ 0	✗	

Figure 5.31: Table of submissions with advanced progress (Stage 3 in flow chart 5.18)

also has started the second review with one defect reported. However, he seems to have forgotten to answer the experience questionnaire which the experiment managers need for the evaluation at the end of this experiment. The user *workshop03* does not seem to have started the experiment yet and might be stuck, this could be the sign for an experiment manager to check if this user is stuck or needs some kind of assistance.

Task Overview (Dashboard) (F15) An important feature for the experiment participants is the task overview or dashboard. This dashboard has the purpose to list all experiments to which the participant is assigned on the main project overview page within the MDRE. After the login into the MDRE, the user can immediately see all his experiments and task which need to be solved by him on the main page.

In figure 5.32, the dashboard that shows all tasks of the example experiment that has been added in this section is displayed. On top of the dashboard, the name of the experiment as well as the start and the end time are visible to the user. Within this time the participants need to complete all tasks of the experiment. The first line within the experiment view shows the description of the experiment which gives the participants additional information. This experiment description is followed by a list of all tasks that need to be completed during the experiment.

The first task in the list called the experience questionnaire is a task in which a Google Forms questionnaire needs to be answered. On the right side of the task, there is a button that links the user to the correct Google Form, so the participant does not need to navigate to any location by himself. The state indicator on the left side indicates the progress that has already been accomplished within the task. In case all state indicators of all tasks show as solved then the participant knows that he has completed all tasks successfully. This feedback is delivered live every few minutes to the user and is an important feature to support the user's progress on the experiment. The second task is a normal task set as a placeholder. This means that there are no submissions possible and therefore there also is no state indicator that shows the progress of the task as there simply is none. In the example, the task is just used to provide extra information to the participants in form of materials that can be downloaded by the users under the specified URL. The third task is a review task in which the participants have to review a model

Workshop: Test Experiment - Start: 27.7.2021 18:00 - End: 27.7.2021 19:00

This is a test experiment

Tasks:
For some tasks it can take a few minutes until the state is updated. Automated refresh happens every 2 minutes, press F5 to reload manually.

- Experience Questionnaire:** Answer the following experience questionnaire via Google Forms. Status: **Unsolved**. Action: [Google Forms](#)
- Download Materials:** Please navigate to the following URL and download all materials needed for this experiment. <https://example.org/>
- Review the Test Model:** Review the model with the name Test Model and report all defects you can find within this model. Status: **Solved**
- Review the Test Model 2:** Review the model with the name Second Test Model and report all defects you can find within this model. Status: **Started**. Action: [Resume Review](#)
- Feedback Questionnaire:** Beantworten Sie das folgende Google Forms Questionnaire bevor Sie fortfahren. Status: **Unsolved**. Action: [Google Forms](#)

Figure 5.32: Task overview (dashboard) for all participants of an experiment (Stage 5 in flow chart 5.18)

based on a predefined review assignment. Submissions are checked in this type of review similar to the submissions of the already explained Google Forms tasks. Once the review is started the state indicator on the left side shows the progress as "started" (eg. task four) and in this example, the review is completed, therefore it displays the progress as "solved". The review itself can either be started by navigating to the assignments via the project details or by simply clicking on the start/resume review button on the right side of the task.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Empirical Evaluation (RQ3)

In this chapter, we present the evaluation of this work. For the evaluation of the tool support components that were added to the MDRE, we conduct a controlled experiment with about 80 participants which is explained in the following section. Finally, the results of our research are presented and put into comparison to provide scientific evidence for our work.

6.1 Study Process

We performed an experimental study to evaluate our research results that are described in this section. The goal of this experiment is the evaluation of the two tool support components for model reviews and experiment administration for the MDRE that have been developed. In detail, this experiment is executed as a controlled experiment in a classroom setting with 78 participants at the Technical University Vienna. The goal of the controlled experiment focuses on the comparison of the advanced MDRE approach (i.e., treatment group) and a traditional pen-and-paper-based review process without tool support (i.e., control group).

In chapter 3, we already presented the most important information about experimentation in software engineering. Within this chapter, we presented the five different phases of an experiment and explained the most important tasks that are executed within them. According to Wohlin et. al. in [78], these five steps include the experiment scoping (E1), planning (E2), operation (E3), analysis (E4), and presentation / packaging (E5). In the following section, we explain the experiment conducted in detail based on these five different phases (see figure 3.1).

6.1.1 Experiment phase - Scoping (E1)

During the experiment scoping the goals and the objects of the study are defined. As already mentioned the main object within this study is the advanced MDRE approach that should be investigated. Within this investigation, the focus is especially on the two new tool support components that have been implemented into the MDRE. The main goal of this experiment is therefore to find out if there are any differences in effectiveness, efficiency, and false-positives between the advanced MDRE approach and the traditional pen-and-paper-based review process.

Variables: In experimentation, we distinguish between two types of variables i.e., independent and dependent variables. Within this experiment, the independent variables include the group assignment and the list of added defects per modeled scenario. The dependent variables are effort, efficiency, effectiveness, reported defects, true defects, and false-positives.

In detail, the *effort* is the amount of time that a participant needed from the beginning to the end of his inspection. *Efficiency* refers to the number of defects that are found within a certain time interval for example defects found per hour. Therefore, the amount of *true defects found* by every participant is divided by the total time in minutes that the participant needed and is then multiplied by 60 minutes. The *effectiveness* is the percentage of how many true defects have been found from the list of total defects that were added to each modeled scenario. Reported defects are simply the amount of all defects that a participant reported. True defects on the other hand are the number of real defects that a participant found out of all reported defects. *False-positive* defects are the percentage of how many false-positives a participant reported from the list of all reported defects.

Hypotheses: The main goal of the implemented tool support components is to improve the whole process for model reviewing. Within the results, we are especially interested in how the performance of the improved MDRE approach is compared to the traditional pen-and-paper-based inspection process. We do not expect the MDRE approach to show an improvement in performance compared to the traditional review method because the MDRE does not automate any parts of the review process itself. The benefit of the MDRE is the organizational and administrative support that is provided which helps reviewers, review managers, and experiment managers. Therefore, we define the following hypotheses for this experiment:

Null hypothesis:

H1.0: There is no difference in the effectiveness when inspecting models of varying types with the advanced MDRE or pen-and-paper-based approach. We do not expect the MDRE approach to have better effectiveness during the model reviews, because the process of the MDRE is strongly inspired by the pen-and-paper-based review approach.

Alternative hypothesis:

H1.1: The advanced MDRE approach is more effective compared to the pen-and-paper review process. The MDRE approach could provide better guidance with the help of different features that have been implemented into the tool support component. One example is a task dashboard that shows the participants which tasks they already solved and which are still unsolved.

Null hypothesis:

H2.0: There is no difference in the efficiency when inspecting models of varying types with the advanced MDRE or pen-and-paper-based approach. We do not expect the MDRE approach to have a better efficiency during the model reviews because of the same reasons we already mentioned for the first null hypothesis.

Alternative hypothesis:

H2.1: The advanced MDRE approach provides a better efficiency compared to the pen-and-paper review process. The MDRE approach could provide better efficiency with the help of the features that we already mentioned in the first alternative hypothesis.

Null hypothesis:

H3.0: There is no difference regarding false-positives when inspecting models of varying types with the advanced MDRE or pen-and-paper-based approach. We do not expect the MDRE approach to result in fewer false-positives during the model reviews because of the same reasons we already mentioned for the first null hypothesis.

Alternative hypothesis:

H3.1: The advanced MDRE approach will report fewer false-positives in comparison to the pen-and-paper-based review process. The MDRE approach could result in fewer false-positives because with the help of the tool support defects in models can be marked more precisely.

During the experiment, feedback is collected to receive insights into the opinion of the participants to find out how they feel about working with the improved MDRE. The feedback also helps to advance the MDRE further and provides detailed information about possible problems with the usability of this application.

6.1.2 Experiment phase - Planning (E2)

During the planning phase of the experiment the study design, materials, the participants, and the execution of the experiment are defined. In general, this experiment is planned as a controlled experiment that is executed in vitro under fully controlled conditions with novice participants doing technology-oriented tasks. Figure 6.1 presents an overview of the experiment design as it is executed by the participants.

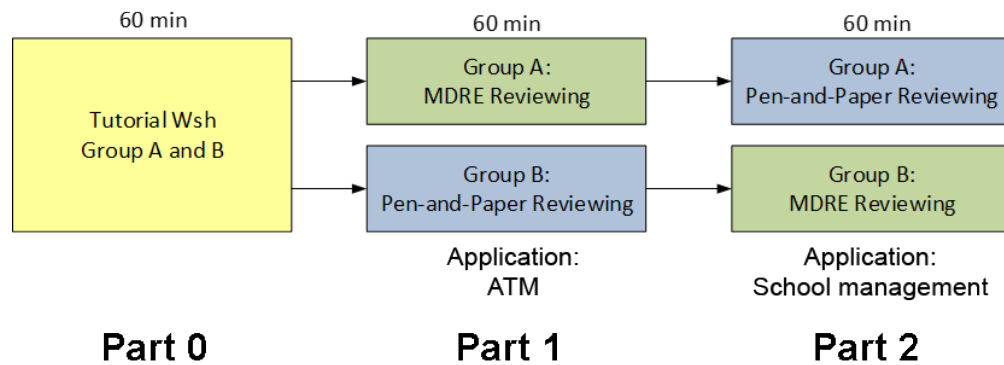


Figure 6.1: Overview of the experiment design in cross-over design

Experiment Design: There are three different phases within this experiment. The first phase is the tutorial phase lasting about 60 minutes. During this time the participants receive a tutorial about the experiment itself. It is explained to them what tasks needed to be performed by them and how the tasks are executed correctly. Another important part of the tutorial is the introduction to the MDRE. None of the participants has worked with the MDRE application before and therefore a tutorial about all features they can or need to use during this experiment is mandatory. In the second part of the tutorial, the main work for the participants starts.

For the experiment execution phase, all participants are split up into two different groups with a focus on different applications (cross-over design). We applied two different application domains that focus on typical scenarios of (a) an ATM and (b) school management processes. Given that there are two different groups it is still an individual work, therefore every participant must work on his own.

Both groups focus on similar models in each part of the experiment, i.e., ATM processes and school management processes. The only difference between the groups is that group A inspects the ATM model with the advanced MDRE approach, while the second group B reviews the same model with the traditional pen-and-paper-based review process. Also within this part, the participants are allowed to work for about 60 minutes.

In the last part of the experiment that also takes 60 minutes the roles change and another model is used. Within this part of the experiment, the model of a school management is used that the participants need to inspect. This time group A, which has performed the last review with the advanced MDRE approach, reviews this model with the traditional pen-and-paper-based review process. Therefore, group B reviews the model in this last part with the advanced MDRE approach. Splitting up all participants of the experiment into these two groups is important to receive a control group that later enables the evaluation of the collected results.

Participants: For this experiment, we recruited 80 participants of which 78 completed the experiment successfully. All participants are undergraduate bachelor students and therefore novice users. The students attended a course on Software Quality Assurance at the Technical University Vienna in the summer term of 2021. During the experiment planning phase, all participants have been split up randomly into two groups to make sure that there is an equal distribution between both groups.

Before the start of the main tasks of the experiment, we collected the experience of each participant using a Google Forms questionnaire¹. This questionnaire evaluated the experience of the participants given the five different categories: *Software Development*, *Model Application*, *Model Creation*, *Model Review*, and *Domain Experience*. Within the

Experience in .. Level	Software Development		Model Application		Model Creation		Model Review		Domain Experience	
	No.	%	No.	%	No.	%	No.	%	No.	%
Less	13	17	14	18	28	36	36	46	13	17
Medium	47	60	48	62	43	55	36	46	32	41
High	18	23	16	20	7	9	6	8	33	42
Total	78	100	78	100	78	100	78	100	78	100

Table 6.1: Experience levels of the participants

questionnaire, the experience was collected based on a 6-point Likert scale that we mapped into three different experience levels which are *less experience*, *medium experience*, and *high experience*. We applied equal importance to all experience categories. Table 6.1 presents the distribution of the experiment participants based on their experience level in the five mentioned categories. The majority of all participants (>50%) could be assigned to the medium or high experience level for all five categories.

Models: Within the experiment, there are two different models, assigned to two different application domains, used during the reviews. To avoid specific limitations because of limited domain knowledge, it is important to use well-known artifacts for the review tasks. Therefore, two application use cases of daily uses have been selected that every participant should be aware of.

The *first application use case* applies a UML state chart for an ATM. We expect that every participant has used an ATM at least once in his life and therefore knows the scenario well. The statechart defines the process of withdrawing or depositing cash at an ATM and consists of 22 different states. We included 28 seeded defects in the statechart that need to be identified by the participants.

The *second application use case* focuses on a UML class diagram of a school management system. Since all participants study Software Engineering at the Technical University Vienna it is safe to assume that every participant has attended a school before and therefore knows the most important entities in the school system. The class diagram

¹Experience questionnaire: forms.gle/19FpPSLQakZdpETeA (last visited 30.09.2021)

defines the most important entities for storing students, parents, teachers, employees, and rooms within a school. This diagram consists of 21 different entities that are defined within it. There are 28 seeded defects in the class diagram which participants have to identify and report during their inspection in the last part of the experiment. All defects inserted into the models were distributed over the usual defect categories that can be observed during the daily use of models. The seeded defects are based on software engineering experience and typical defects in common software engineering projects. These defect categories are *missing*, *wrong*, and *superfluous*. Table 6.2 gives an overview of the distribution of the three defect types within the models. Furthermore, every defect was classified based on its risk using the categories *critical*, *important*, and *minor*.

Defect Type	ATM		School Management	
	No.	%	No.	%
Missing	12	43	12	43
Wrong	12	43	13	46
Superfluous	4	14	3	11
Total	28	100	28	100

Table 6.2: Distribution of seeded defects and defect types within the models

Materials: In addition to the models that are provided to the experiment participants also a bunch of other materials are required to fulfill all tasks in the experiment. The MDRE is also used for the experiment management and not just for the review tool support within this controlled experiment. All files are distributed via the experiment administration component AddOn of the MDRE that has been described within chapter 5. Therefore, for all participants in every group type the *experiment guidelines* and the *model specifications* are provided. Depending on the group of the participants the *models* are either provided as PDF files for pen-and-paper-based groups or directly within the MDRE for the tool-supported groups. The experiment guidelines are defined within a PDF document that is approximately two pages long. Within this document, the most important rules are defined and the whole experiment process from the beginning to the end is explained in detail. These guidelines can be used as a backup in case that the experimentation dashboard of the MDRE would fail. The model specifications are also of importance because they define the whole artifact as it should be displayed within the model. Therefore, the specifications represent the actual state that the model should represent. We consider the specification to be correct. Every state or entity within the model that does not match the specifications within this document precisely is a defect and needs to be reported by the participants.

We used an experience *questionnaire* to capture the background experience of the participants and feedback questionnaires after every experiment part to collect feedback on the applied method. Additionally, the participants of the group that is using the traditional pen-and-paper-based review strategy have another document that is provided to them via the materials. This document is the *defect reporting sheet*, a spreadsheet that

is used by the participants to report defects within the model when they are working with the pen-and-paper-based review process. Within the spreadsheet one line represents one defect and every defect needs to be described and rated based on the severity and the type similar to the defect reporting within the MDRE that was explained in chapter 5. Furthermore, we also record the exact times when a defect is found during the experiment. This provides the possibility to calculate interesting metrics like the time until the first defect was found, found defects per hour, etc. that can be relevant for the evaluation of this experiment. Because the defect reporting is included in the MDRE with the newly added model review component AddOn this spreadsheet is only required for the pen-and-paper-based reviews.

6.1.3 Experiment phase - Operation (E3)

The experiment was executed on two different days to offer the possibility for the participants to choose an appointment that fits best in their weekly schedule. Both appointments for this experiment were conducted in the same week. The first appointment for the experiment has been carried out on Monday, 17th May 2021 between 1 pm and 5 pm, the second appointment took place on Thursday, 20th May 2021 between the same time as the first one. All 80 participants were officially invited to the experiment about two months before the first appointment and had to sign up for their desired appointment. In the end 78 of 80 signed up participants completed all tasks of the experiment. Due to the COVID-19 pandemic, the Technical University Vienna fully switched to distance learning for the summer semester of 2021 and therefore the experiment could not be executed on site but had to be held remotely with the help of online meeting applications. Since the MDRE approach is especially suitable for situations in which people do not work on-site the experiment was never influenced negatively. The operation phase of the experiment is split into three individual sections:

Preparation:

At the start of the experiment, all participants were advised to join a Zoom² online meeting session. The participants needed to keep connected to the online meeting session for the whole duration of the experiment. This online meeting session utilized different meeting rooms. The first room was the main room for communication between the experiment managers and the participants. For the second room, a breakout session was used which the participants could join independently. Within this breakout session, there was the possibility to ask questions to the experiment managers in case there were any problems. Questions in the main room with all participants connected to it would disturb the attention and could interfere with the results of the experiment negatively. Therefore, using a separate room for questions is of importance for the success of the experiment.

²Zoom: www.zoom.us/ (last visited 30.09.2021)

During the preparation for the experiment, the most important details about the experiment were explained to the participants. The experiment managers presented the study design and the main goals of this experiment. In the second step, the new and advanced MDRE approach was explained to the participants. The majority of all participants have never worked with the MDRE before and therefore the MDRE was covered within the experiment tutorial. Furthermore, the participants had the chance to ask questions during and after the tutorial. The preparation for this experiment took about 60 minutes in total. After a short break of about 15 minutes, the participants started with the execution of the experiment and began solving their tasks.

Execution:

In the first step for the execution of this experiment all participants had to log into the MDRE. Since the whole experiment administration is performed with the help of the developed experimentation AddOn for the MDRE both groups (MDRE and pen-and-paper-based review) had to use the MDRE from the beginning. The reason for this is that the tasks for all participants are visible within the MDRE. Figure 6.2 shows the task dashboard of all individual participants during the experiment.

This dashboard was already presented in detail within section 5.3. The participants can see their progress within each task on the left side of the dashboard. After the login, all participants can immediately see their tasks within this experiment. In the first task of the experiment, the experience of the participants was collected. The resulting data is needed for the evaluation of this experiment so that the participants can be classified according to their experience. Therefore, an experience questionnaire was setup with the help of Google Forms. The participants then had to answer this questionnaire before they continued with any later tasks and the MDRE automatically collected the results of the questionnaires and indicated if a questionnaire was answered. Within figure 6.3, the Google Forms experience questionnaire is displayed.

In the next part of the experiment, the review of a UML state chart of a modeled ATM had to be executed by the participants. The participants are split up into two different groups as previously mentioned. Group A conducted the review of the ATM with the advanced MDRE approach. Therefore, the model specifications and the experiment guidelines were provided for this group as additional materials that should help the participants to execute the review. Once the participants of group A downloaded all materials as advised they could simply start the review process with one button click from the experiment dashboard. Group A did not need any kind of spreadsheet for reporting defects within it because the model review AddOn component enables the support for reporting defects within the MDRE. Figure 6.4 shows the defect reporting form of the MDRE which is an important component for the data collection within this experiment. Within section 5.1.3, the process of reporting defects within the MDRE has already been explained in detail.

Group B conducted the same review of the ATM model with the traditional pen-and-

Teil 2: Bankomat: Dauer: ~1h

Review Materialien: Laden Sie die Spezifikation des Beispiels Bankomat und die dazugehörigen Guidelines für das folgende Review herunter. Sie finden diese Materialien unter den folgenden URLs. Spezifikation: <https://tuwel.tuwien.ac.at/mod/resource/view.php?id=1126622> -- Guidelines: <https://tuwel.tuwien.ac.at/mod/resource/view.php?id=1126619>

Review Bankomat - #1: Führen Sie das folgende Review durch und melden Sie so viele Defects wie möglich. Ihre genaue Aufgabe entnehmen Sie bitte aus der zuvor heruntergeladenen Spezifikation und den Guidelines. ▶ Start Review

Review Bankomat - #2: Führen Sie das folgende Review durch und melden Sie so viele Defects wie möglich. Ihre genaue Aufgabe entnehmen Sie bitte aus der zuvor heruntergeladenen Spezifikation und den Guidelines. ▶ Start Review

MDRE Screenshot: Laden Sie die beiden Screenshots aus dem MDRE in welchem alle Ihre verwendeten Marker zu sehen sind, welche Sie zur Markierung der Elemente Ihres Modells verwendet haben, in TUWEL hoch. Link zur Aufgabe: <https://tuwel.tuwien.ac.at/mod/assign/view.php?id=1126634> -- Diese Aufgabe wird händisch bewertet, die Statusänderung kann einige Zeit in Anspruch nehmen.

Figure 6.2: Experiment dashboard shows all tasks that need to be solved

paper-based review approach. For this review, the participants of this group needed more materials than group A. In addition to the experiment guidelines and the model specifications they also needed the model itself and a defect reporting spreadsheet. All materials except the defect reporting sheet were delivered as simple PDF documents for both groups. The defect reporting spreadsheet is the alternative solution for the defect reporting form that is only available for reviews conducted within the MDRE. It is a simple spreadsheet that is available as a Microsoft Excel file and is displayed within figure 6.5. In general, this spreadsheet collects the same data that is also collected by the MDRE defect reporting form.

The participants had about 60 minutes to conduct the review of the ATM within this part of the experiment. At the end of this part, an individual feedback questionnaire had to be answered by all participants via Google Forms. Additionally, materials like the defect reporting spreadsheet, the annotated specification as well as the annotated model were handed in. The form of the questionnaires was similar to the already explained experience questionnaire that has been explained previously. For each of the two groups,

Experience Questionnaire

Your answers will be treated anonymously. Your name will only be used to connect your answers to your inspection records. No individual information will be made public in any form.
Please answer the questions in English.

*** Erforderlich**

matriculation number *

01234567

1. Software Engineering Experience

1.1 In how many software projects did you take an active role? *

0

1-5

5-10

> 10

Figure 6.3: Google Forms experience questionnaire

an individual feedback questionnaire that focuses either on the MDRE approach or on the pen-and-paper-based approach was provided.

After a break of about 15 minutes, the next part of the experiment started. Within the next part of the experiment, another review had to be conducted. The scenario within this part was a UML class diagram that represents the entities and relations of a school management system. For the next part of the experiment, the groups were switched to get representative results from the experiment. Therefore, in this part of the experiment, group A reviewed the model of the school management system with the traditional pen-and-paper-based review process similar to group B in the last part. For this review analogical materials matching the new scenario were provided via the MDRE. The participants solved their tasks after the same method as in the previous part. At the end of this part, the individual feedback questionnaires were answered again by the experiment participants of each group, and materials like the defect reporting sheet, the annotated specification, as well as the annotated model were handed in.

Defect Details ✕

Reported at: 20.05.2021 12:47 **Position:** x: 743.71 y: 711.03 **Width x Height:** 84.00 x 68.00

Defect Description *

Nach der Anzeige Störstoffe entfernen, sollte zusätzlich noch eine Bestätigung mit dem OK Button erfolgen.

Position in Specification (Line number in PDF) *

20

Defect Type * **Defect Severity ***

Missing ⌵

Important ⌵

Close

Figure 6.4: Defect reporting form in the MDRE

Session Attendance:			Defect Severity Class:		
Date:	20.5.2021		Critical (A): The functions affected by these defects are crucial for the customer, i.e., the functions affected are important for the customer and are often used (high severity and risk).		
Starting Time:	14:15		Important (B): The functions affected by these defects are important for the customer, i.e., the functions affected are either important and rarely used or not as important but often used.		
End Time:	15:15		Minor (C): An issue should be changed in the design, but is not an important or crucial defect. An issue is not counted as a defect in the exercise.		
Inspection Record - Bankomat					
ID	Timestamp [hh:mm]	Position in Documents	Defect Type (M, W, U, S)	Defect Severity (A, B, C)	Defect Description
1	14:16	Seite 1, Zeile 1.	M	B	Der Anfangszustand des Bankomats ist im Modell nicht markiert, er sollte aber bei "Warten auf Bankomatkarte" sein!
2	14:19	Seite 1, Zeile 3.	W	A	Zwischen Zustand "Warten auf Bankomatkarte" und "Karte ungültig" sollte in der Bedingung des Übergangs "Karte einführen" "ungültig" und nicht "gültig" stehen.
3	14:21	Seite 1, Zeile 4.	S	C	nach dem Auswerfen der Karte sollte wieder auf eine Bankomatkarte gewartet werden, adas Anzeigen einer Begrüßungsnachricht wird in der Spezifikation nicht erwähnt
4	14:26	Seite 1, Zeile 5.	W	B	Der PIN sollte genau genommen nict beim Verlassen vom Zustand "Warten auf PIN", sondern nach einem Ereignis "PIN Eingabe erfolgt" durchgeführt werden
5	14:28	Seite 1, Zeile 6.	W	A	Der Zähler der Versuche im Zustand "Falscher PIN" sollte beim Eintritt hochgezählt werden (also "++") statt runter ("--")!
6	14:30	Seite 1, Zeile 6.	M	B	Beim Zustand "Falscher PIN" fehlt ein Übergang zurück zum Zustand "Warten auf PIN"
7	14:32	Seite 1, Zeile 7.	M	A	Im Zustand "Bankomatkarte gesperrt" aollte die Karte auch wieder ausgeworfen und nicht nur gesperrt werden!

Figure 6.5: Spreadsheet for defect reporting for pen-and-paper-based reviews

Data collection & Cleanup:

During the whole experiment execution, important data was collected that can be evaluated and turned into results of the study. In detail the following materials were collected during the experiment:

- **Defect reports/Defect reporting spreadsheet:** As already explained all defects were collected with the help of simple defect reports in the MDRE that were persisted in the database for the groups that used the MDRE approach. For the pen-and-paper-based traditional reviews, the defects were collected within a simple spreadsheet that was uploaded by the participants at the end of the experiment. The contents of a defect report have been explained within section 5.1.3.
- **Annotated specification:** The annotated specification is the specification document that has been provided with additional annotations added by the participants. This document is collected to motivate the participants to annotate their specifications to help them to ensure that they do not forget any details.
- **Annotated model:** Similar to the annotated specification the annotated model is the model that has been provided via the MDRE or as a PDF document with annotations added by the participants. The motivation behind the annotated model is the same as with the annotated specification.

Based on the data collected within this experiment the results will be evaluated and presented within section 6.2. At the end of the experiment, a short cleanup session was performed. This session ensures that no participant leaves too early before all tasks were solved and every material was handed in. This cleanup session is supported by the MDRE experimentation AddOn and provides the experiment managers a live overview of the current state of all tasks. Some tasks require the experiment managers to manually tick certain tasks like the hand-in of files as there is no API available that can be used by the MDRE. Other tasks like reviews and Google Form questionnaires are automatically checked by the MDRE and do not require any manual work performed by the experiment managers. The main overview that is provided by the MDRE is called the submission table that was already explained within section 5.3. Within it, the current state of every task of every participant can be checked and altered manually if required.

Figure 6.6 shows the submission table as it was displayed in the early phases of the experiment. It can be seen that at this time the participants of this group already had some progress except one participant that did not attend the experiment in the end. As it can be seen the experiment managers have a good overview of all participants that is updated live and therefore helpful for administrating the experiment. In the second figure 6.7, a later state of the experiment is shown in which one participant already finished the experiment and some others also were close to completing the experiment. This view is also a good indicator for the experiment managers to find out early if a participant forgot to complete some task.

User	Firstname	Lastname	ExpQ	DefRep	AnoSpe	AnoMod	BestQ	RSchul2	RSchul1	MDRESc	AnoSpe	MdreQu
00000001	Andreas	Reich	✓	✓	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000002	Andreas	Reich	✓	✓	✗	✓	✓	✗ 0	✗ 0	✗	✗	✗
00000003	Andreas	Reich	✓	✗	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000004	Andreas	Reich	✓	✓	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000005	Andreas	Reich	✓	✓	✓	✓	✓	✗ 0	✗ 0	✗	✗	✗
00000006	Andreas	Reich	✓	✓	✗	✗	✓	✗ 0	✗ 0	✗	✗	✗
00000007	Andreas	Reich	✓	✓	✓	✗	✓	✗ 0	✗ 0	✗	✗	✗
00000008	Andreas	Reich	✗	✗	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000009	Andreas	Reich	✓	✓	✓	✓	✓	✗ 0	✗ 0	✗	✗	✗
00000010	Andreas	Reich	✓	✓	✓	✓	✓	✗ 0	✗ 0	✗	✗	✗
00000011	Andreas	Reich	✓	✗	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000012	Andreas	Reich	✓	✓	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗

Figure 6.6: Early state of the submission table for experiment managers

User	Firstname	Lastname	ExpQ	DefRep	AnoSpe	AnoMod	BestQ	RSchul2	RSchul1	MDRESc	AnoSpe	MdreQu
00000001	Andreas	Reich	✓	✓	✓	✓	✓	✓ 10	✗ 0	✗	✗	✗
00000002	Andreas	Reich	✓	✓	✓	✓	✓	⊙ 14	✗ 0	✗	✗	✗
00000003	Andreas	Reich	✓	✓	✓	✓	✓	⊙ 14	⊙ 13	✗	✗	✗
00000004	Andreas	Reich	✓	✓	✓	✓	✓	✓ 13	⊙ 7	✗	✗	✗
00000005	Andreas	Reich	✓	✓	✓	✓	✓	✓ 7	⊙ 12	✗	✗	✗
00000006	Andreas	Reich	✓	✓	✓	✓	✓	⊙ 11	⊙ 12	✗	✗	✗
00000007	Andreas	Reich	✓	✓	✓	✗	✓	⊙ 12	✓ 12	✗	✗	✗
00000008	Andreas	Reich	✗	✗	✗	✗	✗	✗ 0	✗ 0	✗	✗	✗
00000009	Andreas	Reich	✓	✓	✓	✓	✓	✓ 12	✓ 12	✓	✓	✓
00000010	Andreas	Reich	✓	✓	✓	✓	✓	✓ 8	⊙ 9	✗	✗	✗
00000011	Andreas	Reich	✓	✓	✓	✓	✓	✓ 12	⊙ 11	✗	✗	✗
00000012	Andreas	Reich	✓	✓	✓	✓	✓	⊙ 11	⊙ 9	✗	✗	✗

Figure 6.7: Later state of the submission table for experiment managers

With the help of the new experiment administration AddOn in the MDRE, it was possible to conduct the whole experiment with less time and resource effort in the administration than in previous experiments. Therefore, the experiment administration proofed its reliableness as there were no problems or errors encountered during the whole experiment. This component also helped us saving time for the participants as the cleanup session was finished in less than 30 minutes and all participants had left the experiment by that time. In previous experiments, the cleanup session took significantly longer because every submission including the Google Forms spreadsheets had to be manually checked by hand and entered into a spreadsheet solution. The experiment administration AddOn automates most of these manual procedures and provides a clean and easy overview in which submissions can be altered manually if required.

6.1.4 Experiment phase - Analysis and Interpretation (E4)

After the experiment operation, a quality check of the collected data was performed and the completeness of the data was assured. This step has been supported by the experimentation AddOn of the MDRE. The analysis and interpretation of the within this experiment collected data and results are performed after the quality check.

The evaluation of the data is performed with the help of different tools. After the experiment is completed, one experiment manager manually maps all reported defects to the defect list of each model in case of a true defect. The experiment management team performs a quality check of the mapped results. Then the data is collected within Microsoft Excel. This data includes the reported defects, true defects, start and end time of all reviews performed by every participant of the experiment. With Microsoft Excel, the effort, efficiency, and effectiveness of the participants within each experiment part are calculated. For the data of every review type, the minimum, maximum, mean, and standard deviation are calculated per experiment part. This data can then be used to compare the different review types with each other.

To test if some of the results are statistically significant the single-sided Mann-Whitney test at a significance level of 95% is used on all results. In case there is any significance in the results the respective null hypothesis can be rejected. In the last step, all collected and calculated data is then exported and imported into R³, a software environment for statistical computation and graphics. In R for all individual data categories, a box plot is generated and exported as an image.

We will present the evaluation of all data and results within section 6.2. In detail, we especially interpret and analyze the collected results based on the dependent variables which are effort, efficiency, effectiveness, reported defects, and false-positives.

³R-Project: www.r-project.org (last visited 30.09.2021)

6.1.5 Experiment phase - Presentation and Package (E5)

The experiment and the analyzed results are packaged and presented within this work. In this chapter, we already presented the study design in detail and explained all information needed to reproduce this experiment. Within section 6.2, we analyze and interpret the collected data and results. Furthermore, the results of the analysis are presented in detail. In the chapters 7 and 8, we discuss and conclude the results from this experiment and we present possible future works that arise from limitations that were discovered during this experiment. A few days after the experiment all participants received feedback telling them the percentage of true defects they found and how their performance within the experiment was compared to the mean of all other participants.

6.2 Results

In this section, we analyze and interpret the results that were collected during the experiment which we conducted to evaluate the outcomes of this work. During the individual parts of the experiment, two different models are used. Therefore, the two parts of the experiment are considered independently of each other. For the statistical evaluation, the single-sided Mann-Whitney test at a significance level of 95% is used.

Effort

The defect detection phase of each part of the experiment is scheduled for up to 60 minutes that the participants can use. All participants had to report their start and end time for the defect detection in each part. While time data has been collected manually for the groups that used the traditional pen-and-paper-based review strategy, the MDRE collected the data automatically without any input needed from the participants. For the evaluation of the effort of this experiment, the reported start and end times are used to calculate the total amount of minutes that the participants have worked on each part of the experiment. The average duration for the review process of the first part in which the modeled ATM state chart was reviewed is 48.5 min (SD: 7.60) for the MDRE and 50.6 min (SD: 8.92) for the traditional pen-and-paper inspection method. In the second part in which the school management class diagram was inspected the average review duration is 45.2 min (SD: 9.33) for the MDRE and 45.1 min (SD: 9.97) for the traditional inspection. The evaluation of the experiment effort does not show any significant differences. In part 1 the MDRE delivers a slightly better result than the traditional inspection, however, we could not confirm this result in part 2. Table 6.3 and the box plot displayed in figure 6.8 summarize the results of the evaluation of the effort that participants have put into solving the tasks of this experiment. The Mann-Whitney test at a significance level of 95% (single-sided) results in the p-value of 0.1325 (-) for part 1 and 0.485 (-) for part 2. Therefore, there are also no significant differences shown by this test.

6. EMPIRICAL EVALUATION (RQ3)

Effort [minutes]		Min	Mean	SD	Max	p-value
Part 1 (ATM)	MDRE	26,5	48,5	7,60	59,6	0,1325 (-)
	Pen and Paper	32,0	50,7	8,92	60,0	
Part 2 (School Management)	MDRE	24,6	45,2	9,33	60,0	0,4850 (-)
	Pen and Paper	27,0	45,1	9,97	60,0	

Table 6.3: Effort of the defect detection [minutes] (MDRE effort automatically captured by the tool support component)

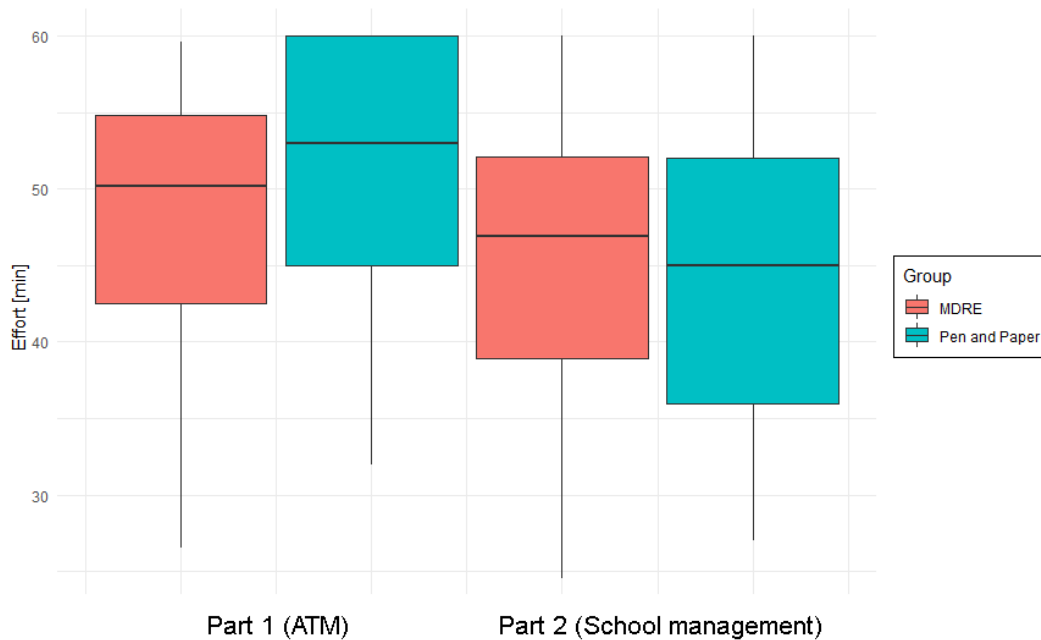


Figure 6.8: Effort of the defect detection [minutes]

Reported Defects

At the end of the experiment, all reported defects of each group and experiment part were collected. The mean values of the number of reported defects for part 1 are 18.7 (SD: 5.28) for the MDRE and 18.0 (SD: 5.66) within the traditional inspection. For part 2 the mean values are 23.7 (SD: 5.57) for the MDRE and 23.7 (SD: 4.20) for the pen-and-paper-based inspection process. Within the first part, there are slightly more defects reported for the MDRE and vice versa for the second part of the experiment. Therefore, there are no significant differences in defect reporting numbers between these two methods. Table 6.4 and the box plot displayed in figure 6.9 summarize the results of the evaluation of the number of reported defects within the two parts of the experiment. Furthermore, applying the Mann-Whitney Test at a significance level of 95% (single-sided) results in the p-value of 0.2623 (-) for part 1 and a p-value of 0.4608 (-) for part 2. This shows that there are also no significant differences measured by this test.

Number of Reported Defects		Min	Mean	SD	Max
Part 1 (ATM)	MDRE	4	18,7	5,28	28
	Pen and Paper	4	18,0	5,66	29
Part 2 (School Management)	MDRE	10	23,7	5,57	35
	Pen and Paper	10	23,7	4,20	30

Table 6.4: Reported defects within the experiment

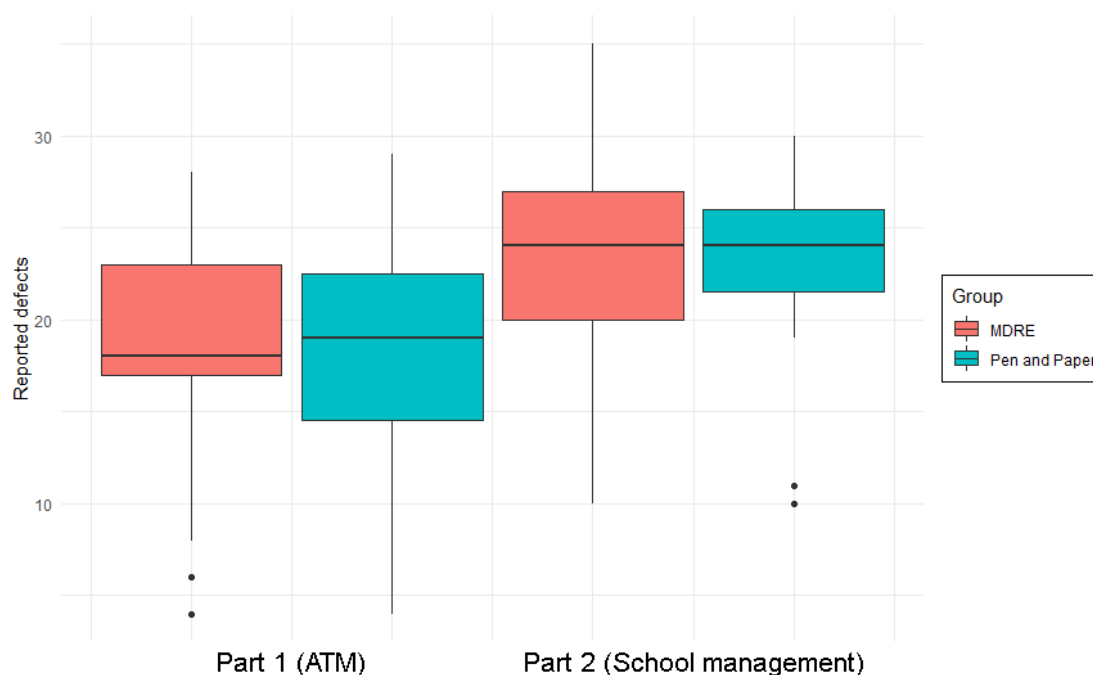


Figure 6.9: Reported defects within the experiment

True Defects

The participants inspected two different models during the controlled experiment. In each of the models, a bunch of defects was added by the experiment managers. To elicit the number of true defects out of the total number of reported defects all reports have been verified and mapped to the list of defects for each model in case of a true defect. The result from this step is the amount of reported true defects of each participant. The mean values of the number of true defects reported for part 1 are 18.2 (SD: 5.17) for the MDRE and 17.3 (SD: 5.43) for the traditional pen-and-paper-based inspection process. For part 2 the mean values are 21.6 (SD: 4.15) for the MDRE and 22.3 (SD: 3.69) for the pen-and-paper-based review approach. Again, the MDRE provides slightly better results than the pen-and-paper-based review approach in part 1 and vice versa in part 2. Therefore, there is no significant difference between these two approaches. Table 6.5 and the box plot displayed in figure 6.10 summarize the results of the evaluation of the

6. EMPIRICAL EVALUATION (RQ3)

number of true defects reported within the two parts of the experiment. Furthermore, applying the Mann-Whitney Test at a significance level of 95% (single-sided) results in the p-value of 0.1839 (-) for part 1 and a p-value of 0.1533 (-) for part 2. This means that also this statistical test does not show any noticeable differences between the two methods.

Number of True Defects		Min	Mean	SD	Max
Part 1 (ATM)	MDRE	2	18,2	5,17	26
	Pen and Paper	4	17,3	5,43	25
Part 2 (School Management)	MDRE	10	21,6	4,15	27
	Pen and Paper	9	22,3	3,69	26

Table 6.5: True defects reported during the experiment

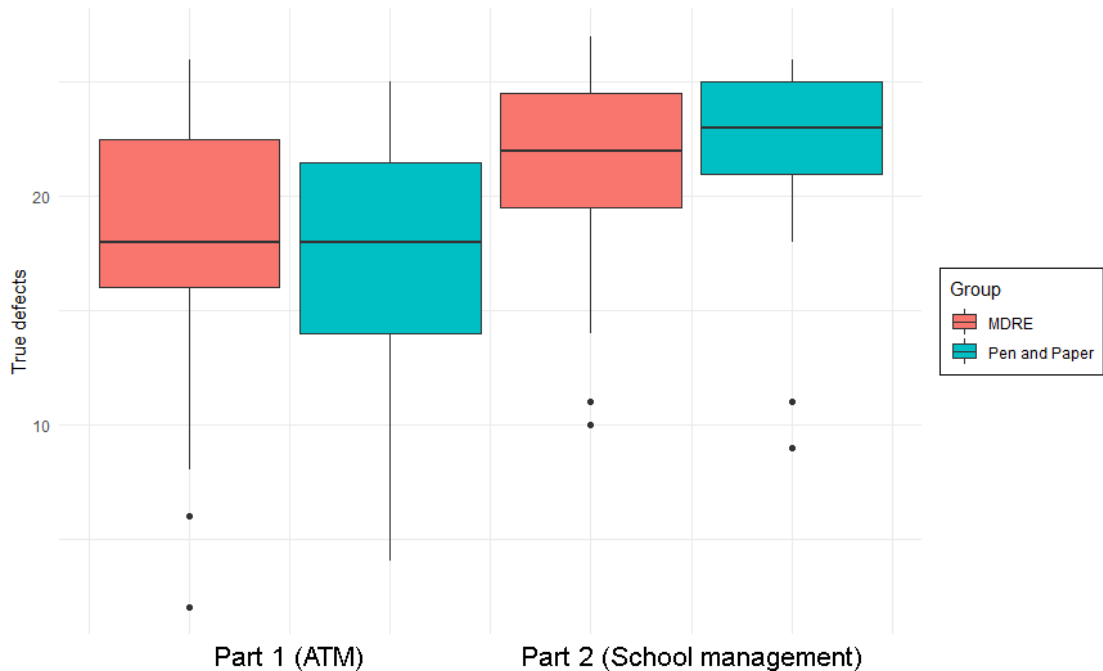


Figure 6.10: True defects reported during the experiment

Effectiveness

The effectiveness of an experiment group that is either using the MDRE or pen-and-paper-based approach is determined by the percentage of true defects found from the total amount of defects per model. In part 1 the mean of the effectiveness is 65.1% (SD: 18.45) for the MDRE and 61.6% (SD: 19.39) for the pen-and-paper-based review process. For part 2 the mean values of the effectiveness are 77.0% (SD: 14.82) for the MDRE and 79.7% (SD: 13.19) for the traditional review approach. Therefore, the MDRE also shows better results in comparison within this category during part 1 and slightly worse results in part 2. The evaluation can not show any significant differences between both approaches. Additionally, the Mann-Whitney Test at a significance level of 95% (single-sided) results in the p-value 0.158 (-) for part 1 and 0.2102 (-) for part 2 and therefore also can not show any significant difference. This means that the hypothesis $H1.0$ (similar defect detection effectiveness), which is defined in section 6.1, cannot be rejected. Table 6.6 and the box plot displayed in figure 6.11 summarize the results of the evaluation of the effectiveness of the experiment.

Effectiveness in %		Min	Mean	SD	Max
Part 1 (ATM)	MDRE	7,1	65,1	18,45	92,9
	Pen and Paper	14,3	61,6	19,39	89,3
Part 2 (School Management)	MDRE	35,7	77,0	14,82	96,4
	Pen and Paper	32,1	79,7	13,19	92,9

Table 6.6: Effectiveness of the experiment [%]

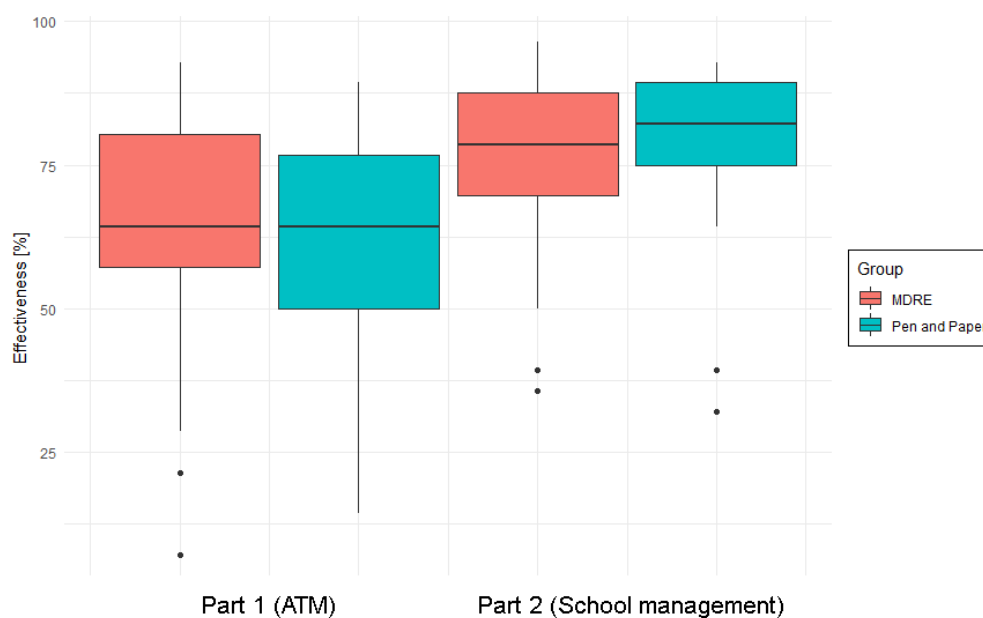


Figure 6.11: Effectiveness of the experiment [%]

Efficiency

The efficiency of a review type is evaluated by the reported number of true defects per time interval (e.g. reported true defects per hour). The mean values of the efficiency for part 1 of this experiment are 22.8 defects per hour (dph) (SD: 7.09) for the MDRE and 21.5 dph (SD: 8.30) for the traditional pen-and-paper-based review process. In part 2 the mean values are 30.0 dph (SD: 9.17) for the MDRE and 31.4 dph (SD: 9.61) for the pen-and-paper-based approach. In both parts, the MDRE approach has slightly less efficiency but the differences are so small that they do not seem to be significant. Furthermore, applying the Mann-Whitney Test at a significance level of 95% (single-sided) resulted in a p-value of 0.2112 (-) for part 1 and a p-value of 0.2772 (-) for part 2. This means that also this statistical test does not show any noticeable differences between the two methods. Therefore, also the hypothesis $H2.0$ (similar defect detection efficiency), which is defined in section 6.1, cannot be rejected. Table 6.7 and the box plot displayed in figure 6.12 summarize the results of the efficiency evaluation of the inspection types used in this experiment.

Efficiency [defects per hour]		Min	Mean	SD	Max	p-value
Part 1 (ATM)	MDRE	4,5	22,8	7,09	38,4	0,2112 (-)
	Pen and Paper	4,0	21,5	8,30	36,0	
Part 2 (School Management)	MDRE	12,6	30,0	9,17	58,6	0,2772 (-)
	Pen and Paper	9,8	31,4	9,61	55,6	

Table 6.7: Efficiency of the experiment [defects per hour]

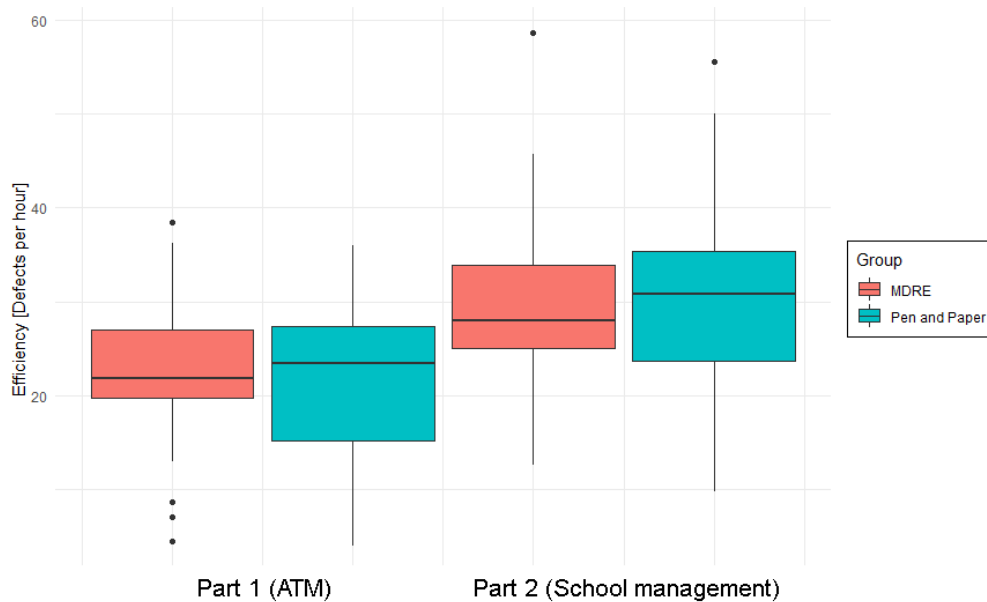


Figure 6.12: Efficiency of the experiment [defects per hour]

False-positives

With the same method as for the evaluation of the number of true defects reported by the participants also the number of false-positives reported can be evaluated. The mean values of the amount of false-positive defects reported within part 1 are 1.7% (SD: 3.12) for the MDRE and 2.6% (SD: 3.79) for the pen-and-paper-based inspection. For part 2 the mean values are 7.5% (SD: 10.90) for the MDRE and 5.1% (SD: 6.61) for the traditional inspection approach. Therefore, the MDRE again shows better results in part 1 and slightly worse in part 2 compared to the pen-and-paper-based inspection. Table 6.8 and the box plot displayed in figure 6.13 summarize the results of the evaluation of the number of false-positive defects reported within the two parts of the experiment. In the box plot, the MDRE shows some bigger statistical outliers in part 2, this could be related to the simplified process for reporting defects compared to the pen-and-paper-based approach. Some participants using the MDRE approach seem to report more defects of which they are unsure if they are true defects in comparison. However, this fact does not need to be negative, because reporting a potential defect that is a false-positive is better than not reporting a defect that turns out to be a true defect. Furthermore, applying the Mann-Whitney Test at a significance level of 95% (single-sided) results in the p-value of 0.1232 (-) for part 1 and a p-value of 0.0603 (-) for part 2. This means that also this statistical test does not show any noticeable differences between the two methods. Therefore, the hypothesis $H3.0$ (similar false-positive detection rates), which is defined in section 6.1, cannot be rejected.

False Positives in %		Min	Mean	SD	Max
Part 1 (ATM)	MDRE	0	1,7	3,12	14,3
	Pen and Paper	0	2,6	3,79	17,9
Part 2 (School Management)	MDRE	0	7,5	10,90	42,9
	Pen and Paper	0	5,1	6,61	21,4

Table 6.8: False-positive reports [%]

6. EMPIRICAL EVALUATION (RQ3)

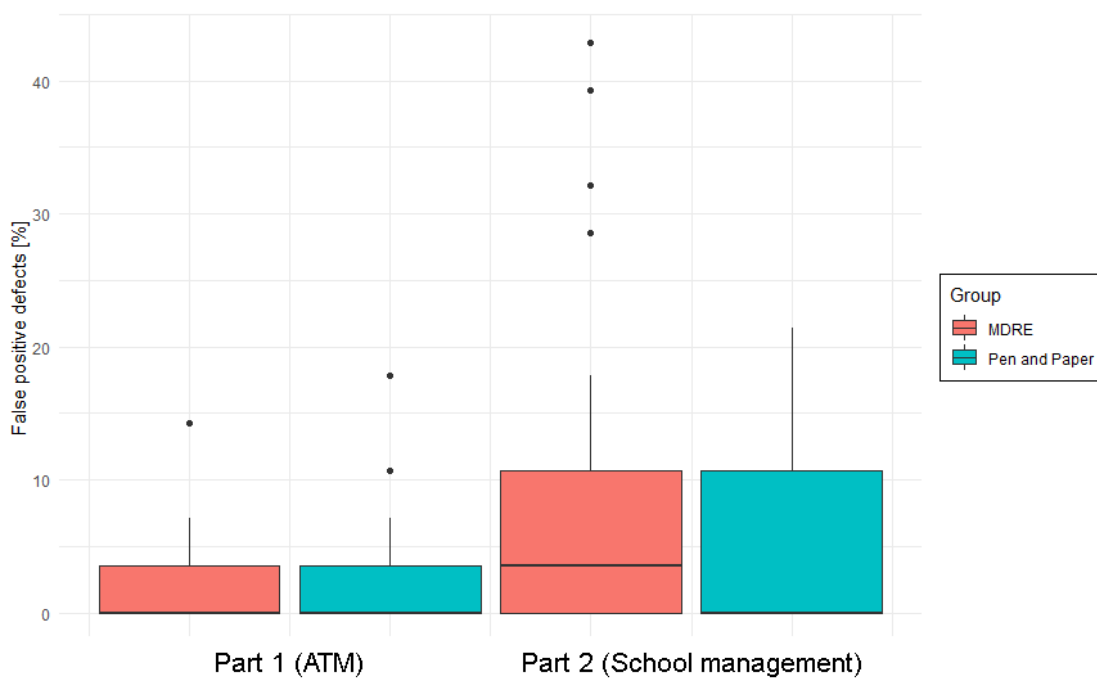


Figure 6.13: False-positive reports [%]

Discussion & Limitation

In this chapter, we discuss the individual results that have been collected and analyzed during the controlled experiment that has been executed. Therefore, every research question is discussed separately during this chapter. Furthermore, the limitations of our work are presented and explained.

7.1 Discussion

During our study, we focused on advancing the currently limited tool support for model reviews, as well as for review and experimentation administration. We used the MDRE in its initial state and extended it with two tool support prototypes. Therefore, we defined and answered the following three research questions within our study:

7.1.1 Prototype for tool-supported model reviewing (RQ1)

Missing tool support made performing reviews time-consuming and generates a lot of administrative overhead. The MDRE provided only basic review capabilities in its initial state. For example, defects had to be reported in the comments section of a review. However, the comment section does not provide the possibility to structure defect reports within it and there is also no way to link a defect report to its location in the model. Another way of reporting reviews with the MDRE would have been a spreadsheet that is filled out using a third-party tool. This solution comes with other pitfalls like increased management effort that is needed for collecting and evaluating all spreadsheets that are handed in. Therefore, we defined the research question:

RQ1: How can model reviews be supported with the help of a model review tool support component?

To solve this research question we implemented a new prototype component for the MDRE that acts as an AddOn and provides improved model review functionalities. With

the help of this component, it is possible to mark the area in which a potential defect is found and to report this area with all details in a unified form. This reduces the overhead in defect reporting and makes it easier for the review managers to process the reported defects. The implemented component is described in detail within section 5.1.3.

The results that have been evaluated within section 6.2 have shown that the implemented tool support component support reviews well. According to the calculated efficiency and effectiveness values, there is no significant difference in using the MDRE compared to the pen-and-paper-based traditional review process. This means that we could confirm the expected results and improve the review capabilities of the MDRE.

7.1.2 Review and Experimentation Process Support (RQ2)

Planning, executing, and administrating reviews and experiments can be laborious with only limited tool support available that could support this process. Our use case that has been defined in chapter 1 is an experiment that compares two different review methods with each other. In this use case, not just the experiment itself must be supported with tool support but also support for conducting reviews must be provided. Therefore, it was one of the main goals of our study to implement and link experiment and review administration together. The initial state of the MDRE did not provide any tool support for experiment administration at all, therefore we defined the research question:

RQ2: How can the administration of reviews and experiments be supported with the help of a tool support component?

To solve this research question we implemented a new prototype component for the MDRE that acts as an AddOn and provides extended support for review and experimentation administration. The component is also connected to the AddOn that was implemented to solve RQ1 and therefore interacts well with the model review tool support. With this new component, the MDRE provides the possibility of creating different tasks and assign them to groups of experiment participants, as well as monitor their progress for each task. This experiment administration AddOn has been explained in detail within section 5.3.

The experiment that has been conducted as a part of the evaluation is described in section 6.1. For this study, the experimentation administration component of the MDRE was used to compare two different review methods with each other. The experiment was a success and all required data could be collected, therefore the AddOn that has been implemented to solve this research question worked well, and also the interaction with the component developed to solve RQ1 worked flawlessly. Especially, the submission table that made it possible to view the state of every participant live was a big advantage along with the support to collect certain submissions (e.g. Google Forms questionnaires) automatically. Winkler et. al. in [77] executed a similar experiment with the MDRE previously to our study and the comparison shows that also the time that is needed for the experiment preparation and the data collection at the end is reduced.

7.1.3 Benefits and limitations of MDRE (RQ3)

It is important to evaluate the benefits and limitations of the implemented tool support. This is especially useful for practitioners who want to introduce the solutions of this study within their industrial use cases. On the other hand, also scientific researchers need the evaluation in case they want to use or extend our work within their research. Therefore, we defined the research question:

RQ3: To what extent are the model review and experiment administration processes improved by the two implemented tool support prototypes?

This evaluation was done with the help of the controlled experiment that has been executed within our study and is presented in chapter 6. Our evaluation shows that with the help of the new tool support components the MDRE is now almost equal in terms of effort, efficiency, and effectiveness compared to the traditional pen-and-paper-based approach. In some categories, the MDRE even shows slightly better results in comparison. To evaluate the performance of the advanced MDRE compared to the pen-and-paper-based traditional approach during the controlled experiment we defined three hypotheses. Within these hypotheses the performance measures of the defect detection effectiveness (H1.0), efficiency (H2.0), and the number of false-positive defects reported (H3.0) are used. During the evaluation of the results from the executed experiment, neither of the three hypotheses could be rejected because we could not find any significant differences between the two approaches. This means that the main goal of our research has been reached and therefore the MDRE can now be used in practice without any disadvantages over the traditional inspection process.

7.2 Limitations

It is important to point out that there are different limitations within our study. During our research we identified the following threats to validity:

Internal validity threats are negative influences that can affect the causality of independent variables. [78] These threats can affect the conclusion about a possible relationship between treatment and outcome. [78] In our research, we used experiment scenarios that are modeled as UML statecharts and UML class diagrams. The experiment results could be different in case there are other models used that are for example more advanced and therefore harder to review for the participants of the experiment. Another internal validity threat of our experiment is the number of defects that have been added to each model by the experiment managers. There was a sum of 56 different defects added into the two models that were reviewed during the experiment. However, there is no guarantee that there are no further defects hidden in the models that have been overlooked by the experiment managers.

External validity threats are conditions that limit the generalization of our experiment results in industrial practice. [78] Our research was conducted with the help of novice users who are all students of a Software Quality Assurance course at the Technical University

Vienna. Therefore, our research results can be different in case the experiment is repeated with domain experts. In general, we are aware of the problems that experiments conducted with students can have. [78] Another possible limitation is the scenarios we used because our participants only consisting out of novice users. The models only consisted out of well-known and simple scenarios like the ATM and the school management in which every user has a good domain experience. Using special industrial scenarios for which the participants do not have any domain experience can affect our results negatively.

Conclusion validity threats affect the ability to draw the correct conclusion about relations that are observed between the treatment and the outcome of an experiment. [78] The previously mentioned external validity threat about using only students within an experiment can also become a conclusion validity threat. In general, the participants in an experiment should be homogeneous to avoid any random variations because of individual differences. However, students at universities even if picked from the same course can be heterogeneous as there is a variation in students. Some students could already be older or they are just working harder than others and therefore have more experience. This means that even using students from another course could affect our results and is, therefore, a threat to the validity.

Construct validity threats are concerns in which the results of the experiment cannot be generalized to the concept or theory behind the experiment. [78] A potential threat to the validity of our results can be that we executed the experiment in a laboratory environment where all variables have been controlled by the experiment managers. The environment in which we executed our experiment cannot be generalized to other environments. Therefore, if this experiment is repeated in a real industrial environment the results could differ from our observations and evaluations.

Conclusion & Future Work

In this chapter, we conclude our work and present the solutions that were achieved within our study. Therefore, we explain our aims again and explain how the developed solutions fulfill the expected results. Furthermore, we present some tasks that could not be implemented within this thesis and can therefore be the base for possible future works.

8.1 Conclusion

The main goal of our research was the improvement of the only limited tool support that was available for reviewing models and administrating reviews and experiments. During our research, we used and advanced the MDRE that had only limited review and administration capabilities in its initial state. To solve our research questions, we aimed for an improvement of the MDRE with the development of two tool support prototypes that were implemented as AddOns into the MDRE. These additional components have the task to provide extended capabilities for model reviews and support the administration of experiments and reviews within the MDRE. The second part of our main goal was the evaluation of the two tool support components that have been added. An evaluation is an important part of our research as it is necessary to prove and validate our results. Evaluated results are important for stakeholders from an industrial area. In case they want to adopt the MDRE within their industrial use cases they need to know the benefits and limitations of the MDRE. But also scientific researchers require a detailed evaluation in case they want to use or extend our research within their studies.

The first component that has been implemented into the MDRE provides additional tool support for model reviews within the MDRE. Before our research started, the initial state of the MDRE did not support defect reporting in a standardized format. Therefore, defects in models had to be reported in plain text with the review comment section of the MDRE, or a third-party application that generates a spreadsheet had to be used. Previously to our work, it was difficult for the review managers to collect all reported

defects within the MDRE and convert them into one single destination format. This task also needed a lot of resources and so the main motivation of our study was to reduce the overhead of model reviewing. With the help of the tool support AddOn for model reviewing it is now possible to report defects in a standardized format within the MDRE by simply drawing a rectangle around the area that contains the defect and add the details to it.

The second component that has been implemented into the MDRE provides extended capabilities for supporting the administration of reviews and experiments. Managing an experiment is a time-consuming task that contains a lot of different steps which need to be executed. Therefore, with the only limited tool support that was available when we started our research the administration of an experiment contained a lot of overhead. With the new tool support component that has been added to MDRE, it is now possible to administrate experiments and reviews faster than before. Especially, the preparation phase before the experiment and the data collection after the experiment can be performed faster and with fewer resources than without tool support. Another important improvement could be achieved in the experiment monitoring. As both implemented components were designed to interact with each other the experiment managers now have the possibility of a live overview that shows the progress of all participants within each task they have to perform.

The expected result of our research is that the MDRE should be advanced so its efficiency and effectiveness when reviewing models are similar to the traditional pen-and-paper-based approach. Within the controlled experiment that was planned and executed to evaluate our research results the effort, efficiency, effectiveness, and the number of true defects found of the MDRE and the pen-and-paper-based approach were collected. The evaluation has proven that with the help of the new prototype components that were implemented into the MDRE all categories for which data was collected do not show any significant differences between the two methods. This means that we achieved the expected results and improved the MDRE to a state in which it is equal to the traditional inspection approach.

Furthermore, we also collected feedback from the experiment participants. Within this feedback, we asked about the experience that the participants had while using the MDRE. Especially, different usability questions are in the central point of focus for this questionnaire. The results of the feedback are displayed in table 8.1 and within the bar chart that is displayed in figure 8.1.

We asked all participants the questions if they needed to learn a lot before using the MDRE, if the MDRE is self-explanatory, if the MDRE is easy to use, and if they would like to use the MDRE in the future again. The results of the feedback show positive results. About 91% of all participants disagreed or strongly disagreed that they needed to learn a lot before they could use the MDRE. Furthermore, 89% of the participants from our experiment agree or strongly agree that the MDRE is self-explanatory. That the MDRE is easy to use is agreed or strongly agreed by 60% of all experiment participants. About 41% of all participants agree or strongly agree to use the MDRE in the future.

Feedback	Need to Learn beforehand		MDRE is self-explain.		MDRE is easy to use		Use MDRE in the future	
	No.	%	No.	%	No.	%	No.	%
Strong Disagree	45	58%	0	0%	3	4%	9	12%
Disagree	26	33%	3	4%	12	15%	10	13%
Neutral	5	6%	6	8%	16	21%	27	35%
Agree	2	3%	13	17%	33	42%	26	33%
Strong Agree	0	0%	56	72%	14	18%	6	8%
Total	78	100%	78	100%	78	100%	78	100%

Table 8.1: MDRE Feedback from experiment participants

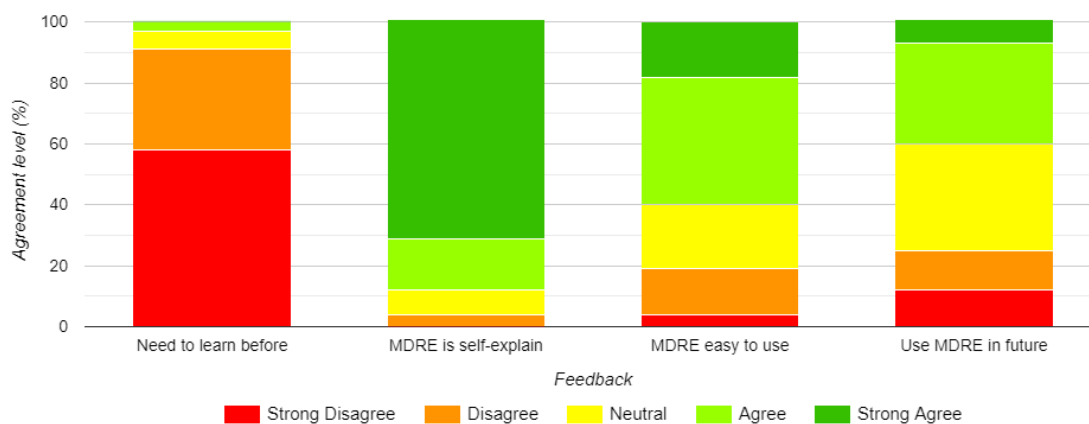


Figure 8.1: MDRE Feedback from experiment participants

However, this is the only category in which there is no clear majority visible in the results. The biggest group that is about 35% answers neutrally to the question if they would like to use the MDRE in the future. In summary, the feedback provided by the experiment participants was overall positive.

8.2 Future Work

Within our research there exist multiple topics that could not be worked on during our study. Therefore, in this section, we present different tasks and topics that can be the basis for possible future works as they would advance the research field further.

The first example for a possible future work is that currently only the area in which a defect is in can be marked within a review in the MDRE. There is no possibility to mark an element itself within a defect report. The highlighting of an area is a solution that is practical but not ideal.

8. CONCLUSION & FUTURE WORK

Another example is the environment in which our controlled experiment was executed. We experimented within a laboratory environment where all parameters have been controlled by the experiment managers. Furthermore, only 78 participants attended and small models were reviewed during the experiment. In a possible future work, scientists could execute the experiment in an industrial environment with larger models and more participants to verify the functionality of the MDRE in different and larger environments.

Furthermore, the MDRE currently does not support any kind of team reviews, this means that all reviewers only can work on their own at the moment. Due to the COVID-19 pandemic many people are working remotely nowadays and will continue to work from home in the future. Therefore, performing team reviews could become an important requirement for the MDRE in the future.

List of Figures

1.1	Reviewing and empirical study process challenges	4
1.2	Overview of the work packages of the solution approach	6
2.1	Steps in the SDLC [63]	10
2.2	Phases of the Waterfall Model [6]	12
2.3	Phases of the Spiral Model [58]	13
2.4	Phases of the Scrum Model [60]	15
2.5	Dimensions and sub-dimensions of software inspections [38]	22
2.6	Code review in Review Board	26
2.7	Code review in Gerrit	28
2.8	Code review in Atlassian Crucible	29
2.9	Pull requests in GitHub	31
3.1	Phases of an experiment [78]	40
4.1	Design science research cycles [23]	46
5.1	Architecture of the MDRE	54
5.2	Simplified class diagram of the initial MDRE	58
5.3	Overview of projects	59
5.4	Audit Logging	60
5.5	List of model configurations	61
5.6	Model design view	61
5.7	List of review assignments	62
5.8	Model review view [77]	63
5.9	MDRE Review execution flow chart	66
5.10	Class diagram MDRE - model review AddOn	66
5.11	Review view of the MDRE	67
5.12	Highlighting a defect within the review view of the MDRE	68
5.13	Defect reporting form	70
5.14	Defect reporting form with example input	70
5.15	List of reported defects and review finalization	71
5.16	Dialog to save and close a review	72
5.17	Rejected review in the review assignment overview	73
5.18	MDRE Review execution flow chart	75

5.19	Class diagram MDRE - experiment and review administration AddOn . . .	76
5.20	Form to add an experiment	78
5.21	Overview of all experiments	79
5.22	Experiment participants overview	80
5.23	Form to add a normal task in an experiment	82
5.24	Form to add a review task in an experiment	83
5.25	Form to add a Google Forms task in an experiment	84
5.26	Overview of all tasks in an experiment	84
5.27	Adding / editing of a submission	85
5.28	Overview of all submissions in a task	86
5.29	Table of submissions within an experiment	87
5.30	Manual submission control within the submission table	87
5.31	Table of submissions with advanced progress	88
5.32	Task overview (dashboard) for all participants of an experiment	89
6.1	Overview of the experiment design in cross-over design	94
6.2	Experiment dashboard shows all tasks that need to be solved	99
6.3	Google Forms experience questionnaire	100
6.4	Defect reporting form in the MDRE	101
6.5	Spreadsheet for defect reporting for pen-and-paper-based reviews	101
6.6	Early state of the submission table for experiment managers	103
6.7	Later state of the submission table for experiment managers	103
6.8	Effort of the defect detection [minutes]	106
6.9	Reported defects within the experiment	107
6.10	True defects reported during the experiment	108
6.11	Effectiveness of the experiment [%]	109
6.12	Efficiency of the experiment [defects per hour]	110
6.13	False-positive reports [%]	112
8.1	MDRE Feedback from experiment participants	119

List of Tables

2.1	Comparison of advantages and disadvantages of the inspection tools	32
5.1	Comparison of the MDRE with well-known software inspection tools	51
5.2	Features of the MDRE in its initial state [56]	53
5.3	Feature level comparison of the MDRE (after model review AddOn)	65
5.4	Feature level comparison of the MDRE (after experimentation AddOn) . .	77
6.1	Experience levels of the participants	95
6.2	Distribution of seeded defects and defect types within the models	96
6.3	Effort of the defect detection	106
6.4	Reported defects within the experiment	107
6.5	True defects reported during the experiment	108
6.6	Effectiveness of the experiment [%]	109
6.7	Efficiency of the experiment [defects per hour]	110
6.8	False-positive reports [%]	111
8.1	MDRE Feedback from experiment participants	119



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

- [1] C. Anderson. Docker [software engineering]. *IEEE Software*, 32(3):102–c3, 2015. doi:10.1109/MS.2015.62.
- [2] C. Atkinson and T. Kuhne. Model-driven development: a metamodeling foundation. *IEEE Software*, 20(5):36–41, 2003. doi:10.1109/MS.2003.1231149.
- [3] K. Banker, P. Bakkum, and T. Hawkins. *MongoDB in Action*. Manning Publications, 2016.
- [4] V. R. Basili. The role of experimentation in software engineering: past, current, and future. In *Proceedings of IEEE 18th International Conference on Software Engineering*, pages 442–449, 1996. doi:10.1109/ICSE.1996.493439.
- [5] V. R. Basili, R. W. Selby, and D. H. Hutchens. Experimentation in software engineering. *IEEE Transactions on Software Engineering*, SE-12(7):733–743, 1986. doi:10.1109/TSE.1986.6312975.
- [6] Y. Bassil. A simulation model for the waterfall software development life cycle. *CoRR*, abs/1205.6904, 2012. URL: <http://arxiv.org/abs/1205.6904>, arXiv:1205.6904.
- [7] J. Bézivin. *Model Driven Engineering: An Emerging Technical Space*, pages 36–64. Springer Berlin Heidelberg, Berlin, Heidelberg, 2006. doi:10.1007/11877028_2.
- [8] A. Bosu and J. C. Carver. Peer code review in open source communities using reviewboard. In *Proceedings of the ACM 4th Annual Workshop on Evaluation and Usability of Programming Languages and Tools, PLATEAU '12*, page 17–24, New York, NY, USA, 2012. Association for Computing Machinery. doi:10.1145/2414721.2414726.
- [9] M. Brambilla, J. Cabot, and M. Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool, 2012.
- [10] A. W. Brown, J. Conallen, and D. Tropeano. *Introduction: Models, Modeling, and Model-Driven Architecture (MDA)*, pages 1–16. Springer Berlin Heidelberg, Berlin, Heidelberg, 2005. doi:10.1007/3-540-28554-7_1.

- [11] F. J. Buckley and R. Poston. Software quality assurance. *IEEE Transactions on Software Engineering*, SE-10(1):36–41, 1984. doi:10.1109/TSE.1984.5010196.
- [12] M. Butcher. *Mastering OpenLDAP: Configuring, Securing and Integrating Directory Services*. Packt Publishing, 2007.
- [13] M. Chemuturi. *Mastering Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers*. J. Ross Publishing, Inc., 2010.
- [14] B. Curtis. Measurement and experimentation in software engineering. *Proceedings of the IEEE*, 68(9):1144–1157, 1980. doi:10.1109/PROC.1980.11813.
- [15] A. Davis, E. Bersoff, and E. Comer. A strategy for comparing alternative software development life cycle models. *IEEE Trans. Software Eng.*, 14:1453–1461, 1988.
- [16] M. Dawson, D. Burrell, E. Rahim, and S. Brewster. Integrating software assurance into the software development life cycle (sdic). *Journal of Information Systems Technology and Planning*, 3:49–53, 01 2010.
- [17] A. Dresch, D. P. Lacerda, and J. A. V. Antunes. *Design Science Research*. 2014. doi:10.1007/978-3-319-07374-3.
- [18] E. Engström, M. Storey, P. Runeson, M. Höst, and M. Baldassarre. How software engineering research aligns with design science: a review. *Empirical Software Engineering*, 25, 07 2020. doi:10.1007/s10664-020-09818-7.
- [19] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE '07)*, pages 37–54, 2007. doi:10.1109/FOSE.2007.14.
- [20] J. E. Gaffney. Metrics in software quality assurance. In *Proceedings of the ACM '81 Conference*, ACM 81, page 126–130, New York, NY, USA, 1981. Association for Computing Machinery. doi:10.1145/800175.809854.
- [21] D. Gašević, D. Djuric, and V. Devedžić. *Model Driven Engineering*, pages 125–155. Springer Berlin Heidelberg, Berlin, Heidelberg, 2009. doi:10.1007/978-3-642-00282-3_4.
- [22] M. Halling, S. Biffl, T. Grechenig, and M. Kohle. Using reading techniques to focus inspection performance. In *Proceedings 27th EUROMICRO Conference. 2001: A Net Odyssey*, pages 248–257, 2001. doi:10.1109/EURMIC.2001.952461.
- [23] A. Hevner and S. Chatterjee. *Design Research in Information Systems: Theory and Practice*. Springer US, Boston, MA, 2010. doi:10.1007/978-1-4419-5653-8_1.
- [24] A. Hevner, S. T. March, J. Park, and S. Ram. Design science in information systems research. *Management Information Systems Quarterly*, 28:75–, 03 2004.

- [25] J. Hutchinson, M. Rouncefield, and J. Whittle. Model-driven engineering practices in industry. In *Proceedings of the 33rd International Conference on Software Engineering, ICSE '11*, page 633–642, New York, NY, USA, 2011. Association for Computing Machinery. doi:10.1145/1985793.1985882.
- [26] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. *Reporting Experiments in Software Engineering*, pages 201–228. Springer London, London, 2008. doi:10.1007/978-1-84800-044-5_8.
- [27] A. Jedlitschka, N. Juristo, and D. Rombach. Reporting experiments to satisfy professionals' information needs. *Empirical Software Engineering*, 19(6):1921–1955, 2014. doi:10.1007/s10664-013-9268-6.
- [28] D. Jemerov and S. Isakova. *Kotlin in Action*. Manning Publications, New York, 2017.
- [29] P. Johannesson and E. Perjons. *An Introduction to Design Science*. 07 2014. doi:10.1007/978-3-319-10632-8.
- [30] M. Jones, B. Campbell, and C. Mortimore. Json web token (jwt) profile for oauth 2.0 client authentication and authorization grants. Available: <https://tools.ietf.org/html/rfc7523>, 05 2015.
- [31] S. Juba, A. Vannahme, and A. Volkov. *Learning PostgreSQL*. Packt Publishing, 11 2015.
- [32] N. Juristo and A. M. Moreno. *Basics of Software Engineering Experimentation*. Springer Publishing Company, Incorporated, 1st edition, 2010. doi:10.5555/1965114.
- [33] M. Kalinowski and G. H. Travassos. Ispis: A framework supporting software inspection processes. pages 392–393, 01 2004. doi:10.1109/ASE.2004.1342772.
- [34] M. Kalinowski and G. H. Travassos. Ispis: From conception towards industry readiness. In *XXVI International Conference of the Chilean Society of Computer Science (SCCC'07)*, pages 132–141, 2007. doi:10.1109/SCCC.2007.9.
- [35] A. Kalyan, M. Chiam, J. Sun, and S. Manoharan. A collaborative code review platform for github. pages 191–196, 11 2016. doi:10.1109/ICECCS.2016.032.
- [36] H. Koziolok. The role of experimentation in software engineering. In *Seminar "Research Methods", Summer Term*. Citeseer, 2005.
- [37] S. Kumar and P. Dubey. Software developmnt life cycle (sdlc) analytical comparison and survey on traditional and agile methodology. *Abhinav National Monthly Refereed Journal of Research in Science and Technology*, 2:22–30, 08 2013.

- [38] O. Laitenberger. A survey of software inspection technologies. 08 2001. doi:10.1142/9789812389701_0023.
- [39] O. Laitenberger and J. DeBaud. Encompassing life cycle centric survey of software inspection. *Journal of Systems and Software*, 50:5–31, 2000. doi:10.1016/S0164-1212(99)00073-4.
- [40] C. Laporte and A. April. *Software Quality Assurance*. 12 2017. doi:10.1002/97811119312451.
- [41] Y. Leau, W. K. Loo, W. Y. Tham, and S. F. Tan. Software development life cycle agile vs traditional approaches. volume 37, 02 2012.
- [42] A. Lucia, F. Ferrucci, and G. Tortora. *Emerging Methods, Technologies and Process Management in Software Engineering*. 03 2008. doi:10.1002/9780470238103.
- [43] S. March and V. Storey. Design science in the information systems discipline: An introduction to the special issue on design science research. *MIS Quarterly*, 32, 12 2008. doi:10.2307/25148869.
- [44] L. Milanese. *Learning Gerrit Code Review*. Community experience distilled. Packt Publishing, 2013.
- [45] A. Mishra. A comparative study of different software development life cycle models in different scenarios. *International Journal of Advance Research in Computer Science and Management Studies*, 1:64–69, 10 2013.
- [46] P. Mohagheghi and J. Aagedal. Evaluating quality in model-driven engineering. In *International Workshop on Modeling in Software Engineering (MISE'07: ICSE Workshop 2007)*, pages 6–6, 2007. doi:10.1109/MISE.2007.6.
- [47] S. Naik. *Software Testing and Quality Assurance*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2007. doi:10.1002/9780470382844.
- [48] N. Nurmuliani, D. Zowghi, and S. Powell. Analysis of requirements volatility during software development life cycle. In *2004 Australian Software Engineering Conference. Proceedings.*, pages 28–37, 2004. doi:10.1109/ASWEC.2004.1290455.
- [49] D. L. Parnas and M. Lawford. The role of inspection in software quality assurance. *IEEE Transactions on Software Engineering*, 29(8):674–676, 2003. doi:10.1109/TSE.2003.1223642.
- [50] O. Pastor, S. España, J. I. Panach, and N. Aquino. Model-driven development. *Informatik-Spektrum*, 31(5):394–407, Oct 2008. doi:10.1007/s00287-008-0275-8.
- [51] K. Peffers, T. Tuunanen, M. A. Rothenberger, and S. Chatterjee. A design science research methodology for information systems research. *Journal of Management Information Systems*, 24(3):45–77, 2007. doi:10.2753/MIS0742-1222240302.

- [52] P. Permien. Distributed code-reviews using gerrit, 2012.
- [53] S. L. Pfleeger. Experimental design and analysis in software engineering. *Annals of Software Engineering*, 1(1):219–253, Dec 1995. doi:10.1007/BF02249052.
- [54] S. L. Pfleeger. Experimentation in software engineering. *Adv. Comput.*, 44:127–167, 1997.
- [55] S. Prabakaran and T. Bhuvaneshwari. A survey on software development life cycle models. *International Journal of Computer Science and Mobile Computing*, 05 2013.
- [56] A. Prock, C. Engelbrecht, Isikoglu M., C. Burger, and D. Kretz. Model design and review editor: Featureliste. Technical report cdl-sqi 2021-12, TU Wien, Vienna, Austria, September 2021.
- [57] S. Rawat. *Getting Started with Review Board*. Packt Publishing, 2014.
- [58] N. B. Ruparelia. Software development lifecycle models. *SIGSOFT Softw. Eng. Notes*, 35(3):8–13, May 2010. doi:10.1145/1764810.1764814.
- [59] M. Sabou, D. Winkler, and S. Biffl. *Empirical Software Engineering Experimentation with Human Computation*, pages 173–215. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-32489-6_7.
- [60] C. Schmidt. *Agile Software Development*, pages 7–35. Springer International Publishing, Cham, 2016. doi:10.1007/978-3-319-26057-0_2.
- [61] B. Selic. The pragmatics of model-driven development. *IEEE Software*, 20(5):19–25, 2003. doi:10.1109/MS.2003.1231146.
- [62] B. Selic. Model-driven development: its essence and opportunities. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC'06)*, page 7 pp., 2006. doi:10.1109/ISORC.2006.54.
- [63] S. Shylesh. A study of software development life cycle process models. *Social Science Research Network*, 2017. doi:10.2139/ssrn.2988291.
- [64] D. I. K. Sjoeborg, J. E. Hannay, O. Hansen, V. B. Kampenes, A. Karahasanovic, N.-K. Liborg, and A. C. Rekdal. A survey of controlled experiments in software engineering. *IEEE Transactions on Software Engineering*, 31(9):733–753, 2005. doi:10.1109/TSE.2005.97.
- [65] D. Sjøberg, B. Anda, E. Arisholm, T. Dybå, M. Jørgensen, A. Karahasanovic, E. Koren, and M. Vokác. Conducting realistic experiments in software engineering. pages 17 – 26, 02 2002. doi:10.1109/ISESE.2002.1166921.
- [66] K.-J. Stol and B. Fitzgerald. *Guidelines for Conducting Software Engineering Research*, pages 27–62. Springer International Publishing, Cham, 2020. doi:10.1007/978-3-030-32489-6_2.

- [67] O. Söderström and E. Moradian. Secure audit log management. *Procedia Computer Science*, 22:1249–1258, 2013. 17th International Conference in Knowledge Based and Intelligent Information and Engineering Systems - KES2013. doi:10.1016/j.procs.2013.09.212.
- [68] T. Thelin, P. Runeson, and C. Wohlin. An experimental comparison of usage-based and checklist-based reading. *IEEE Transactions on Software Engineering*, 29(8):687–704, 2003. doi:10.1109/TSE.2003.1223644.
- [69] J. Tian. *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. 02 2005.
- [70] Y. Tymchuk, A. Mocci, and M. Lanza. Code review: Veni, vidi, vici. In *2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, pages 151–160, 2015. doi:10.1109/SANER.2015.7081825.
- [71] V. K. Vaishnavi and Jr. W. Kuechler. *Design Science Research Methods and Patterns: Innovating Information and Communication Technology*. Auerbach Publications, USA, 1st edition, 2007.
- [72] N. Walkinshaw. *Software Quality Assurance: Consistency in the Face of Complexity and Change*. 01 2017. doi:10.1007/978-3-319-64822-4.
- [73] J. Whittle, J. Hutchinson, and M. Rouncefield. The state of practice in model-driven engineering. *IEEE Software*, 31(3):79–85, 2014. doi:10.1109/MS.2013.65.
- [74] R. J. Wieringa. Design science methodology: principles and practice. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, volume 2, pages 493–494, 2010. doi:10.1145/1810295.1810446.
- [75] R. J. Wieringa. *Design science methodology for information systems and software engineering*. Springer, 2014. doi:10.1007/978-3-662-43839-8.
- [76] D. Winkler, S. Biffl, and B. Thurnher. Investigating the impact of active guidance on design inspection. volume 3547, pages 117–163, 06 2005. doi:10.1007/11497455_36.
- [77] D. Winkler, K. Meixner, D. Kretz, A. Zweckstetter, and S. Biffl. A controlled experiment on distributed model reviewing with tool support. Technical report cdl-sqi 2021-06, TU Wien, Vienna, Austria, May 2021. submitted to ESEM 2021.
- [78] C. Wohlin, P. Runeson, M. Host, M. C. Ohlsson, B. Regnell, and A. Wesslen. *Experimentation in software engineering*. Springer Science and Business Media, 2012. doi:10.1007/978-3-642-29044-2.
- [79] C. Yinyi, Z. Kefa, and W. Jinlin. Performance analysis of postgresql and mongodb databases for unstructured data. In *Proceedings of the 2019 International Conference on Mathematics, Big Data Analysis and Simulation and Modelling (MBDASM 2019)*, pages 60–62. Atlantis Press, 10 2019. doi:10.2991/mbdasm-19.2019.14.

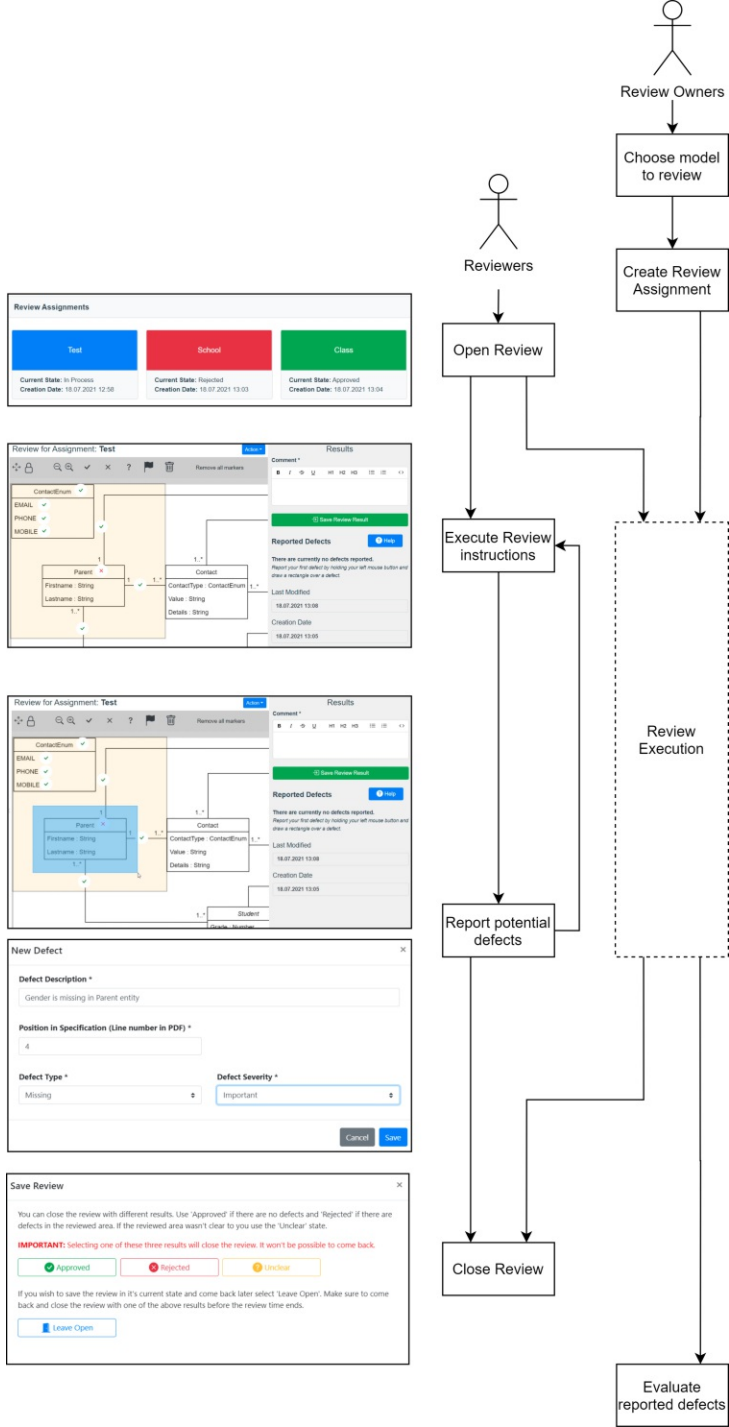
- [80] Y. Yu. Reviewer recommender of pull-request in github. 09 2014. doi:10.1109/ICSME.2014.107.
- [81] Y. Zhu. *Software Reading Techniques: Twenty Techniques for More Effective Software Review and Inspection*. Apress, Berkeley, CA, 2016. doi:10.1007/978-1-4842-2346-8.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Appendices

Appendix A: Flow chart MDRE review execution



Appendix B: Flow chart MDRE experiment administration

