Gottfried Wilhelm
Leibniz Universität Hannover
Faculty of Electrical Engineering
and Computer Science
Institute of Practical Computer Science
Software Engineering Group

Extending Use Case Tables with Explainability Needs

Bachelor's Thesis

in Computer Science

by

Denise Behrens

First Examiner: Prof. Dr. Kurt Schneider
Second Examiner: Dr. Jil Klünder
Supervisor: Jakob Droste, M. Sc.

Hanover, March 22nd 2024

Declaration of Independence

I hereby certify that I have written this Bachelor's Thesis independently and without outside help and that I have not used any sources or aids other than those specified in the work. The work has not yet been submitted to any other examination office in the same or similar form.

Hanover, March 22nd 2024

Denise Behrens

Acknowledgement

First and foremost, I am very grateful to my husband Kersten. His support, patience, and understanding were instrumental in helping me navigate through the challenges of this academic endeavor. His constant encouragement and belief in my abilities provided the strength I needed to persevere during the writing process and throughout my entire studies.

I extend my appreciation to my supervisor, Jakob Droste, for his invaluable guidance, expertise, and constructive feedback.

I would also like to thank my family and friends for their unconditional support and encouragement. Their belief in me, coupled with their willingness to lend a helping hand whenever needed, has been a source of motivation throughout this journey.

Abstract

Satisfying non-functional requirements, such as explainability, is becoming increasingly important in the development of new software to improve user understanding and software quality. Explanations can help users who have difficulty using a software. Additionally they can explain the purpose and increase the understanding of certain User Interface (UI) elements. In order to make the development process more efficient, the need for explainability should be identified and noted during the planning phase. Possible difficulties in understanding on the part of the future user group can thus be identified early on and taken into account directly during development and testing.

This thesis documents the development process of the 'Use Case Tool', including requirements elicitation and documentation, aimed at capturing explainability needs early on in the software development process. The tool parses all Use Case tables of a Software Requirements Specification (SRS) in copyable PDF format and displays them in a UI. Each step of the Use Cases' main scenario and extensions can be extended with explainability needs. The extended Use Case tables can be exported as PDF or CSV files and reimported into the tool.

To validate the usability of the tool, usability tests with the System Usability Scale (SUS) were conducted with nine participants. The average SUS score was 87.8%, indicating excellent usability. In addition, comments and reactions of the attendees were noted and used to further improve the 'Use Case Tool'. The results show that the features of the developed tool are well integrated and that the system was self-explanatory for the study participants.

Zusammenfassung

Die Erfüllung nicht-funktionaler Anforderungen, wie z. B. Erklärbarkeit, wird bei der Entwicklung neuer Software immer wichtiger, um das Verständnis der Benutzer und die Qualität der Software zu verbessern. Erklärungen können Benutzern helfen, die Schwierigkeiten bei der Benutzung einer Software haben. Darüber hinaus können sie den Zweck erklären und das Verständnis bestimmter Benutzeroberflächen-Elemente verbessern. Um den Entwicklungsprozess effizienter zu gestalten, sollte der Bedarf an Erklärbarkeit bereits in der Planungsphase erkannt und berücksichtigt werden. Mögliche Verständnisschwierigkeiten der zukünftigen Nutzergruppe können so frühzeitig erkannt und direkt bei der Entwicklung und beim Testen berücksichtigt werden.

Die vorliegende Arbeit dokumentiert den Entwicklungsprozess des 'Use Case Tools', einschließlich der Anforderungserhebung und -dokumentation, zur frühzeitigen Erfassung von Erklärungsbedarf im Softwareentwicklungsprozess. Das Tool analysiert alle Use Case-Tabellen einer Software Requirements Specification (SRS) im kopierfähigen PDF-Format und stellt sie in einer Benutzeroberfläche dar. Jeder Schritt des Hauptszenarios und der Erweiterungen der Use Cases kann um Erklärungsbedarf erweitert werden. Die erweiterten Use Case-Tabellen können als PDF- oder CSV-Dateien exportiert und wieder in das Tool importiert werden.

Um die Benutzerfreundlichkeit des Tools zu validieren, wurden Usability-Tests mit der System Usability Scale (SUS) mit neun Teilnehmern durchgeführt. Der durchschnittliche SUS-Wert lag bei 87,8%, was auf eine ausgezeichnete Benutzerfreundlichkeit hindeutet. Darüber hinaus wurden die Kommentare und Reaktionen der Teilnehmer notiert und zur weiteren Verbesserung des 'Use Case Tools' genutzt. Die Ergebnisse zeigen, dass die Funktionen des entwickelten Tools gut integriert sind und dass das System für die Studienteilnehmer selbsterklärend war.

Contents

Chapter 1

Introduction

Modern software systems are increasingly characterized by the use of artificial intelligence and complex algorithms. This means that software is becoming increasingly opaque and incomprehensible to its users [16, 35, 36], which can lead to a decreasing trust that users have in a software [35]. To avoid this problem, developers can integrate explanations into the software. Explanations are intended to help users operate the system or clarify certain issues for them [16].

1.1 Motivation

Explainability, as a Non-Functional Requirement (NFR), is becoming increasingly important in the development of new software systems [16, 35]. Satisfying NFRs is important for achieving high software quality [15]. One difficulty is to find out at an early stage where there might be ambiguities in the use of the software for the future user group. The first step towards developing a new software is planning. In traditional development processes, this usually involves writing a Software Requirements Specification (SRS), which was first defined by the Institute of Electrical and Electronic Engineers (IEEE) in 1984 [25]. An SRS contains the customer's requirements for the software and describes the planned implementation. To get a more accurate picture of the software's functionality, Use Cases are written for each interaction between the system and its actors [29]. This is usually done in the form of tables, for example based on the template introduced by Cockburn [17]. When reviewing the Use Case tables, software stakeholders may already notice ambiguities in the system.

1.2 Problem Statement

The development of new software faces a significant challenge: users cannot test it until the development or prototyping phase is complete. Consequently, the software may lack clarity or explanations, or fail to convey its intended benefits to users. However, ensuring user satisfaction with the level of explainability prior to implementation is challenging, as users often have to rely on their tacit knowledge [19]. Currently, explainability needs are addressed after implementation, requiring iterative development cycles to improve them. Users can only provide feedback based on their experience with the software that has already been implemented, which can then be taken into account to start with a new iterative development cycle. These iterative processes increase development time and cost. This underlines the importance of identifying explainability requirements early in the design phase, as noted by Chazette et al. [14].

1.3 Solution Approach

The aim of this work is to integrate additional explainability requirements into software systems as early as possible. The best time to intervene is during software planning, after the SRS is written. In this thesis, a tool to extend Use Case tables from an SRS with explainability needs is developed. The tool can import SRSs in copyable PDF format, from which the Use Case tables are extracted and displayed. Explanations can be added to the steps of the main scenario and extensions in the User Interface (UI), which are displayed in an additional row in the tables when the Use Case tables are exported. This way, stakeholders that review the Use Case tables can express their explainability needs at an early stage in development, which in turn can be taken into account when implementing the software. This makes the integration of explanations more efficient. Ultimately, it makes the software easier to understand and more attractive to the user. In order to test the usability of the developed tool, usability tests are carried out with nine participants. The tool is being developed for the Software Engineering Group of Leibniz University, which focuses on planned and structured software development. Their aim is to use the tool for future studies.

1.4 Thesis Structure

To achieve the above goals, this thesis first discusses the basics of SRS, explainability, and technical prerequisites in Chapter 2. It also provides a classification of related work. Chapter 3 covers the requirements elicitation for the tool to be developed with functional, non-functional and explainability requirements. This is followed by the description of the implementation in Chapter 4, after which the structure of the study is discussed in Chapter 5. The results of the usability testing are presented in Chapter 6. The discussion of the results, improvements and limitations of this work are described in Chapter 7. Finally, Chapter 8 concludes and discusses future work.

Chapter 2

Background and Related Work

This chapter introduces basic software engineering concepts and standards of software development that are relevant to the understanding of this work. This is followed by a discussion of related work on parsing and processing SRS content.

2.1 Software Requirements Specification

In 1984, the IEEE specified what should be written in a good Software Requirements Specification (SRS), and what characteristics it should have [25]. The SRS, as the name implies, is about the collection of requirements from a customer for a new software to be developed. To maintain flexibility, these requirements should avoid details about design, project management, or verification. Only design constraints should be used when necessary. In addition, all requirements should be ranked to define which requirement is the most important and which is just an add-on [25].

The SRS should be the product of both the customer's and the supplier's expertise. The customer is an expert in his or her field and is aware of the requirements for the new software. The supplier is an expert in his or her field and therefore knows everything about the development process and software design. Only their combined knowledge can lead to a good SRS [25, 26].

According to IEEE [25], a good SRS should be unambiguous, complete, verifiable, consistent, modifiable, traceable and usable during operation and maintenance phase. This means that all requirements should have only one interpretation, and if the SRS is written in natural language, it should be checked for ambiguity by someone independent [26]. In order to be complete, an SRS must include functional and non-functional requirements, as well as necessary constraints or interfaces [25, 26]. All functions and features that a software has or should have are expressed in functional requirements. NFRs are also known as quality requirements or constraints. This means that the software is restricted to a certain extent. For example, a certain level of performance, security, or maintainability may be required, so lower-level requirements are removed from the solution [11, 28]. In addition, it is necessary to define how the software will react to a variety of valid and invalid input data. A requirement is verifiable if it is possible to measure whether it has succeeded or failed. To ensure consistency and modifiability in an SRS, requirements should not conflict, duplication should be avoided, and changes should be easily possible. Traceability is required in an SRS so that the various stages of development can be traced backwards and forwards [25, 26].

2.1.1 Use Cases

Use cases are used within an SRS to clarify how the software being developed will behave based on its functionality. A Use Case describes the interaction between an actor, who wants to achieve a certain goal, and the software [18, 23]. Use cases should be written as simple text to avoid misunderstandings between the supplier and the customer. In addition, Use Cases are used as a contract between all parties [18].

In 1987, Ivar Jacobson first introduced the idea of Use Cases at the OOPSLA (Object-Oriented Programming, Systems, Languages, and Applications) conference [29]. He realized that not all of a system's behaviors could easily be written down in a single description. Consequently, it was more convenient to describe different behaviors separately in order to be understandable. One of these behaviors, which is further divided into a dialogue between the user and the system, is a Use Case. The dialogue is a combination of a user action and a system response. At this point, it should be clarified that a user is meant to be a person who actually interacts with the system. In contrast, a user can have different roles. Each role has different tasks while using the system. A Use Case ends when no more (new) actions can be performed. Then another Use Case can be triggered [29].

In general, there is not just one right way to write Use Cases. It always depends on what the goal is that someone is trying to accomplish with a Use Case. For example, Use Cases can be used to gather requirements, discuss about future requirements, describe a work process, or create documentation for the system's design. When a new system is being developed, black box Use Cases are written. White box Use Cases are written when the purpose is a work process. Black box means that there is no need to know the inner workings of a system, while white box means that it is necessary to know everything about the inner workings [18].

In Table 2.1, a template for a Use Case table is shown. It is based on the template and work by Alistair Cockburn [17, 18] and is currently the template used by the Software Engineering Group. The only difference from this table is that the actual template is in German. The first row contains the Use Case number and a brief description of the goal that the Use Case reflects. This is followed by a longer explanation of the goal of the Use Case in its context. The scope describes (part of) the system referred to in the Use Case, and the level describes the importance of the task or function. Preconditions define the state of the system or world that the Use Case expects. Until the state is not completed, the system has a state called the minimal guarantee. The success guarantee is the state when the goal is achieved. Stakeholders are people who are directly or indirectly involved in a Use Case. A primary actor is a person or group with the same role who primarily gets involved with the Use Case. A trigger is an action in the system that causes a Use Case to be activated [17, 18]. The main scenario [30] or main success scenario [17, 18] describes the dialogue between the user and the system in steps, starting from the trigger and ending with the goal achieved or the cleanup afterwards. The extensions contain all scenarios that cause a different action to that of the main scenario, always referring to one step. Priority and frequency are marked as optional by Cockburn, but are included in the Software Engineering Group's template. Frequency indicates how often this Use Case will occur, and priority reflects whether this is an important feature [17, 18].

Use Case <1>	**<goal as a short active verb phrase>**
Goal in Context	<a longer statement of the goal in context if needed>
Scope	<what system is being considered black box under design>
Level	<one of: Summary, Primary Task, Subfunction>
Preconditions	<what we expect is already the state of the world>
Minimal Guarantee	<the state of the world upon successful completion>
Success Guarantee	<the state of the world if goal abandoned>
Stakeholders	<other systems relied upon to accomplish Use Case>
Primary actor	<a role name or description for the primary actor>
Trigger	<the action upon the system that starts the Use Case>
Main scenario	**<Step> <Action>** 1. <put here the steps of the scenario from trigger to goal delivery, and any cleanup after> 2. < ... >
Extensions	**<Step> <Branching Action>** 1a. <condition causing branching> : <action or name of sub.Use Case> 1b. < ... >
Priority	<how critical to your system / organization>
Frequency	<how often it is expected to happen>

Table 2.1: Use Case table template according to Cockburn [17, 18], adapted for use by the Software Engineering Group

In order to get an overview and a better understanding of the relationships between all Use Cases of a system, a Use Case diagram can be drawn. This is typically done using the Unified Modeling Language (UML) [18, 30]. An example is shown in Figure 2.1, which depicts Use Cases of the software developed in this thesis. Actors are drawn as stick figures and Use Cases as ellipses. The frame is the system border, which shows everything that is happening in the system. There are several arrows in UML. In the figure, there are arrows with an «include» from one Use Case to another. This means that the Use Case where the line begins is dependent on the Use Case to which the arrow points. An arrow with «extend» points from one Use Case to another, interrupting the Use Case from which it comes [18, 23]. Other systems interacting with the Use Cases are drawn as rectangles [23].

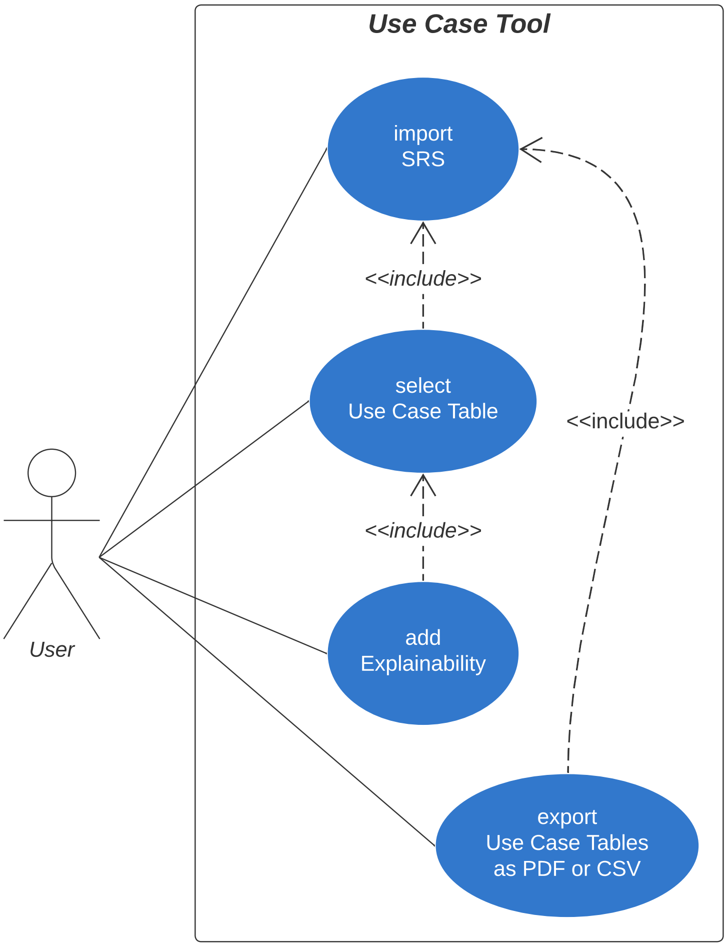


Figure 2.1: Example of a Use Case diagram (depicting the Use Cases of the software developed in this thesis)

2.1.2 Acceptance Testing

Acceptance tests are defined at the end of an SRS and acceptance testing follows the implementation of a system. This is to ensure that the previously defined Use Cases are fulfilled [27, 29]. Each acceptance test is written with a setup, user inputs, and corresponding desired outputs. Typically, the input is an action a user performs on the system, and the output is the expected response or behaviour of the system [41]. The fact that the acceptance tests are based on the Use Cases, and therefore the customer's requirements, makes it easy for the customer to determine whether the system is acceptable or not. [41]. In general, acceptance testing is used to validate the functionality of a developed system. This includes checking the interactions between the user and the system, as well as the constraints and interfaces [24]. Examples of acceptance tests are the ones for the software developed in this thesis, which can be found in Appendix A.

2.2 Explainability

Deters et al. [20] propose the following definition of explainability: "Explainability is the ability of a software to be explained to an addressee, given a specific context of use and depending on the goals of the explainer." In the context of software systems, the addressee to whom the system is explained is a stakeholder of the software. Explanations within this context typically originate from the software system itself, making the system self-explanatory. Here, the goal is not dictated by the explainer, but rather by the product owner who has established the explainability requirements for the software [20]. In a self-explanatory system, explanations should be integrated into the UI in a meaningful and appropriate way. This means adding information that the user needs to understand, for example, the system's features, but also omitting unnecessary information. In addition, the design choices of the chosen explanations are important to avoid overwhelming the user with too much information and to avoid unnecessarily increasing other NFRs such as resource consumption [16]. Explainability is a NFR [16, 35]. It is linked to and may interfere with other NFRs such as transparency and understandability [14, 20], usability [14, 16], performance, development cost, and security [35]. Therefore, it is important to think about the possible benefits or mismatches of other NFRs with explainability when doing requirements analysis [16].

2.3 Likert Scale

In 1932, Rensis Likert invented the Likert scale, which is named after him [37]. Likert scales are used to measure people's agreement with statements [31, 37, 46, 49]. It is important to note that according to Likert, the statements should be "expressions of desired behavior and not statements of fact" [37]. This is because a statement of fact leads to a response not with the person's current attitude toward the statement, but most likely with a preconceived attitude. In addition, all statements should be clear and unambiguous so that there is no confusion in the reader's mind. Likert defined that it is beneficial to have half of the questions' answers (e.g. the odd ones) on one side of the agreement level and the other half (e.g. the even ones) on the other side [37]. This helps to prevent participants from giving habitual answers or from losing concentration as they have to think about the given statements [12, 37].

To create a questionnaire, several statements are collected. There should be more statements than the number that will end up in the final questionnaire, because they will then be tested on people who are representative of the future test group and sorted out [12, 37, 46]. After that, the statements that are the most appropriate for the purpose of the questionnaire will be selected [37]. The level of agreement of each participant is originally shown in a 5- or 7-point scale [12, 37, 49]. Table 2.2 shows an example of a 5-point scale where each attitude is assigned to a number (e.g. Agree = 4) in order to be able to calculate a score [37].

1	**2**	**3**	**4**	**5**
Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree

Table 2.2: 5-point Likert responding format [13, 37, 49]

2.4 System Usability Scale

The System Usability Scale (SUS) is a Likert scale and was developed by John Brooke in 1996 [12]. It is designed to quickly and easily measure the usability of a system using subjective sentiment statements [1, 12]. Ten statements were selected from a pool of fifty statements based on the fact that they produced the most extreme responses when rated on a 5-point Likert scale for two systems requiring different levels of knowledge. Table 2.3 shows the ten selected statements which are given to the participants after they have tested a system. The odd-numbered statements received a higher level of agreement when tested with the pool of fifty statements, and the even-numbered statements received a higher level of disagreement. The selection of statements from a pool and the arrangement of the statements indicate that the SUS is a Likert scale, as described in Section 2.3. Within the SUS, the level of agreement is typically measured with a 5-point Likert scale, as shown in Table 2.2 [12].

Number	Statement
1	I think that I would like to use this system frequently
2	I found the system unnecessarily complex
3	I thought the system was easy to use
4	I think that I would need the support of a technical person to be able to use this system
5	I found the various functions in this system were well integrated
6	I thought there was too much inconsistency in this system
7	I would imagine that most people would learn to use this system very quickly
8	I found the system very cumbersome to use
9	I felt very confident using the system
10	I needed to learn a lot of things before I could get going with this system

Table 2.3: Brooke's SUS Statements [12]

After the SUS is performed, the score should be determined. For each odd-numbered statement (1, 3, 5, 7, 9), subtract 1 from the attitude value. For example, if statement one (Table 2.3) was answered with 'Agree' (=4), the calculation based on the ranking in Table 2.2, is '4' minus 1, resulting in a score of 3 for statement one. For each even-numbered statement (2, 4, 6, 8, 10), subtract the attitude value from 5. For example, if statement two (Table 2.3) was answered with 'Strongly Disagree' (=1), the calculation is 5 minus '1', resulting in a score of 4. After doing this for all 10 statements, sum the results for each respondent and multiply each sum by 2.5 to get the System Usability (SU). The SU ranges from 0 to 100 [12]. A higher score means a better usability, as shown in Figure 2.2. Better usability leads to, but does not guarantee, higher acceptance according to Bangor et al. [1].

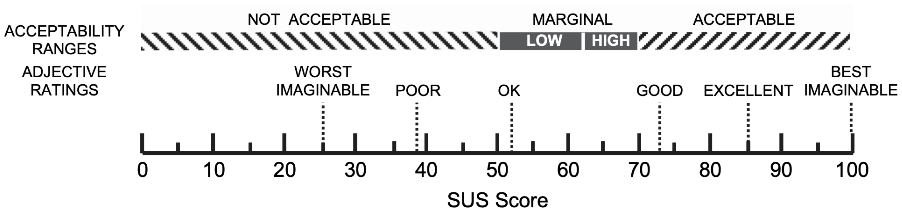


Figure 2.2: SUS score and it's acceptance, by Bangor et al. [1]

2.5 Extensible Application Markup Language

The eXtensible Application Markup Language (XAML) was developed by Microsoft for its .NET framework and is based on the eXtensible Markup Language (XML). It is a declarative language to describe UIs. Each XAML file contains a file of associated code – the *codebehind*. This includes handlers or events whose behaviors need to be implemented [38]. However, it should be carefully considered whether it makes sense to write the *codebehind* directly into the XAML dependent file, or whether it is better to outsource it to a separate class that is not UI related. One approach could be applying the Model-View-ViewModel (MVVM) pattern [48], which is described in Section 2.7.

2.6 Avalonia

Avalonia is a UI framework launched in late 2013 under the name Perspex by Steven Kirk [8, 32]. Kirk's plan was to develop a XAML-based UI framework on an open source basis. He wanted to improve the known bugs and problems of the existing Windows Presentation Foundation (WPF) UI framework [32]. Another of Kirk's goals was to make the framework run on a variety of platforms, and not just on Windows like WPF does [6, 32]. Today, applications developed with Avalonia can be used on Windows, MacOS, Linux, Android, iOS, and WebAssembly. A special feature is that the UI looks and feels identical on all platforms [6, 8]. These approaches allow developers to develop on a preferred platform for a customer's desired platform. In addition, Avalonia enables independent rendering which gives even more flexibility and customization to its users [6]. As an open source project hosted on GitHub [7], Avalonia is constantly being improved and reviewed [6]. Users can ask questions and provide feedback on existing developments. This way, the community working on the framework is able to get continuous feedback to develop new features or improve existing ones. Users are free to add their own features by forking the Avalonia GitHub project and adding their own code. Since Avalonia is licensed under the MIT license [7], it is free of charge and can be commercially used.

2.7 Model-View-ViewModel Pattern

The Model-View-ViewModel (MVVM) pattern helps to separate the code for the UI from the business logic of an application. This allows more freedom for changes on both sides. It also simplifies testing, maintainability, and further development. Different developers can work simultaneously on UI design and business logic [48]. This can limit the cost and time spent on the application. In addition, the MVVM pattern makes it easier to reuse parts of the code, whether it is the design or the business logic [48].

Figure 2.3 depicts the connections between the View, the ViewModel, and the Model. The solid lines from right to left show that the View has knowledge about the ViewModel, and the ViewModel has knowledge about the Model. This knowledge does not flow in the other direction. The View is the UI that the user sees when using the software, typically written in XAML. It should contain as little *codebehind* as possible. The ViewModel is and should be used to define properties and commands that are invoked in the View through data binding. The dashed line from the ViewModel to the View shows that the ViewModel sends notifications to the View when a state changes. In the Model, the data model and the business logic are defined. It is updated by the ViewModel and sends notifications to the ViewModel when a state changes [48].

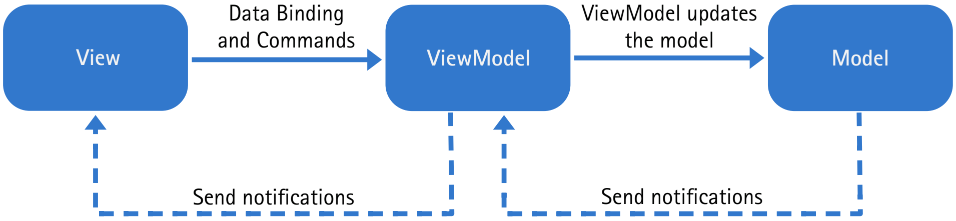


Figure 2.3: Model-View-ViewModel pattern according to Stonis [48]

2.8 Tabula

Manuel Aristarán started developing Tabula in late 2012 [4] and presented it to the public in an article in early 2013 [3]. Tabula was developed as a web application. It was designed to upload text based PDF files and extract table data from them. The extracted data was in CSV format. Since Tabula is licensed under the MIT license, it is open source and free of charge [3]. In the middle of 2013, Aristarán et al. [4] began working on the Tabula repository on GitHub. They updated and developed the repository for several years with the help of their users and sponsors who could help them with bug reports, feature ideas, and financial support. The repository has not been updated since 2020, but there are now other GitHub repositories that use Tabula as a base, for example to embed it in a C# project [10].

2.9 Related Work

Several papers have been found that have the goal of improving the quality of SRSs and thus also improving the result of the final product of an SRS.

Since this is also part of the goal of this thesis, these papers can be considered related work.

Osborne and MacNish [43] presented a method to enhance the clarity of SRSs by using controlled natural language in conjunction with Natural Language Processing (NLP) techniques in 1996. Their goal was to resolve ambiguities in SRS by limiting the language scope and extending the capabilities of a toolkit with features such as parse selection and error diagnosis. Their case study demonstrates how their system can refine SRS assertions, detect ambiguities, and generate logical forms to improve the alignment with customer needs and reduce system errors.

Since 1996, other authors such as Umber and Bajwa [52], Kuchta and Padhiyar [34], or Fatwanto [22] have also worked on decreasing ambiguities of natural language in SRSs with different approaches. Umber and Bajwa [52] proposed a method to transform an English SRS into a controlled representation using the Semantic of Business Vocabulary and Rules (SBVR) standard. Kuchta and Padhiyar [34] developed an experimental application to extract disambiguating SRS concepts by calculating distances between meanings derived from the WordNet ontology. Fatwanto [22] proposed a method to reduce the ambiguity and incompleteness of natural language SRSs by transforming them into object-oriented ones.

Ali et al. [2] developed a methodology to enhance the quality of SRSs, addressing the unsatisfactory success rate of software products due to SRS deficiencies. Their approach described steps to ensure the completeness and correctness of requirements, including parsing for domain comprehensiveness, stakeholder perspective mapping, validation, and an external review. The result of this process is the Total Quality Score for the tested SRS. The use of the authors' method in projects resulted in a better score than the score with external review only.

All these papers work on the same goal: to achieve high software quality based on an SRS. Most of the papers found include the ambiguities that come with the usage of natural written language. This thesis also focuses to a certain extent on ambiguities in the Use Case tables of an SRS, but more specifically on noting missing explanations that the future user group needs in order to understand and use the software to be developed.

Chapter 3

Requirements Elicitation

This chapter deals with the requirements elicitation process and the stakeholders of the ‘Use Case Tool’. Functional and non-functional requirements are determined and a guideline for the development of the tool is created. Since the application is being developed for the Software Engineering Group of the Leibniz University, the following requirements are elicited from the supervisor of this thesis.

3.1 Stakeholders

Stakeholders can be divided into primary and secondary stakeholders [39]. Primary stakeholders have a direct impact on the outcome of the product, for example, they have ordered the product and have an overview of the finances. They can be private sector companies, local or national government agencies, as well as influential figures like politicians and officials who may not be directly affected but have the ability to influence outcomes. Secondary stakeholders do not have direct influence, but they have an interest in the outcome of a product. They may be customers who will use the product, or employees who fear losing their jobs because of the product [39].

For the ‘Use Case Tool’, a primary stakeholder is the Software Engineering Group of Leibniz University, as they initiated the development of the tool. In addition, the developer has significant influence over the outcome of the tool and is therefore another primary stakeholder. Other primary stakeholders are the Leibniz University, the state Lower Saxony and Germany, as they indirectly influence the research of the Software Engineering Group through funding [21].

The users for whom the ‘Use Case Tool’ was developed are considered secondary stakeholders. These are mainly study participants with knowledge about explainability needs who extend Use Case tables from an SRS. Secondary stakeholders are also those involved in the usability testing of the tool. Since secondary stakeholders are considered to be German, the language used within the tool is German. In the following, the Software Engineering Group, a primary stakeholder, and all secondary stakeholders are referred to as users because they will be using the tool for development, testing, studies, and research.

3.2 Functional Requirements

All necessary functions and features of the 'Use Case Tool' are collected and prioritized. The first requirement has the highest priority and the last the lowest.

R01 The tool should be able to import Use Case tables following the Cockburn template [17] from an SRS according to the template of the Software Engineering Group in copyable PDF format.

R02 All Use Case table titles should be clearly displayed in the UI and be clickable in order to access the corresponding Use Case table.

R03 Each individual step of the main scenario and extensions should be able to be enriched with explainability needs in the tool's UI.

R04 The explainability needs added to the main scenario and extensions steps are added to the Use Case tables in a new line called "Erklärungsbedarf" and the corresponding step number when the Use Case tables are exported.

R05 The tool should be able to export extended Use Case tables as PDF and CSV files.

The most important functional requirement of the 'Use Case Tool' is the ability to import the Use Case tables from an SRS that is in a copyable PDF format (R01). The SRS template is defined by the Engineering Group and the Use Case tables included are based on the template by Cockburn [17]. The next important functional requirement is that all Use Case tables should be clearly displayed in the UI (R02). This is necessary to navigate through all the contents of the tables. The main scenario and extensions steps of each Use Case should be able to be enriched with explainability needs in the UI (R03). This is the main purpose of the tool to be developed, but it is not the highest priority in the functional requirements, because the first requirements have to be developed before this can be implemented. The next functional requirement is that the explainability needs added in the UI can be added to Use Case tables as an additional row called "Erklärungsbedarf" (explainability needs) when exported (R04). It is important that the new row should contain the step that was extended in the main scenario or in the extensions. The least important functional requirement is the export of the extended Use Case tables as PDF and CSV files (R05). It should be noted that the importance of the functional requirements was also prioritized after the implementation process.

3.3 Non-functional Requirements

This section contains the NFRs, which are some constraints to the 'Use Case Tool'. They are also prioritized from highest to lowest priority.

NR01 The tool should be easy to install on a system running Windows 11.

NR02 The tool should be usable on a screen of at least 16 inches in size.

NR03 The UI should be in German.

Most importantly, since Windows 11 is the default operating system of the Software Engineering Group's computers, the 'Use Case Tool' should be easy to install and run on a system running Windows 11 (NR01). The next important NFR is that the tool should be usable on a screen of at least 16 inches in size, so that there is enough space to display the contents of the Use Case tables in an appropriate font size (NR02), and that no UI elements are truncated. Since the tool is intended for German developers and users, the UI should be in German (NR03). This is the least important requirement, since the tool itself will not contain much text, so it can be changed quickly.

3.4 Workflow

Once the requirements were established, the workflow of the 'Use Case Tool' was defined to ensure that all the necessary functionality was considered. The workflow is shown as an activity diagram in Figure 3.1. There are two partitions for the actors which are the user and the tool itself. The black dot represents the starting point of the activity after opening the 'Use Case Tool'. First, the user can import an SRS that is in a copyable PDF format. Then, the tool parses the Use Case tables from the SRS and splits the main scenario and the extensions into steps. Next, the user can select a Use Case table from a list of all tables. If the user wants to add explainability needs, an input field belonging to a step of the main scenario or the extensions can be used. The user input will be written to the internal data structure of the tool. Afterwards, the user can select and edit other Use Case tables or decide to export the extended Use Case tables as PDF or CSV file. If the user wants to add more explainability needs to another Use Case table, another table can be selected and the aforementioned steps are repeated. When the user finishes adding explainability needs and exports the Use Case tables, the tool adds a row called "Erklärungsbedarf" to the extended Use Case tables and exports all tables. This completes the activity, which is indicated by the black dot surrounded by another circle.

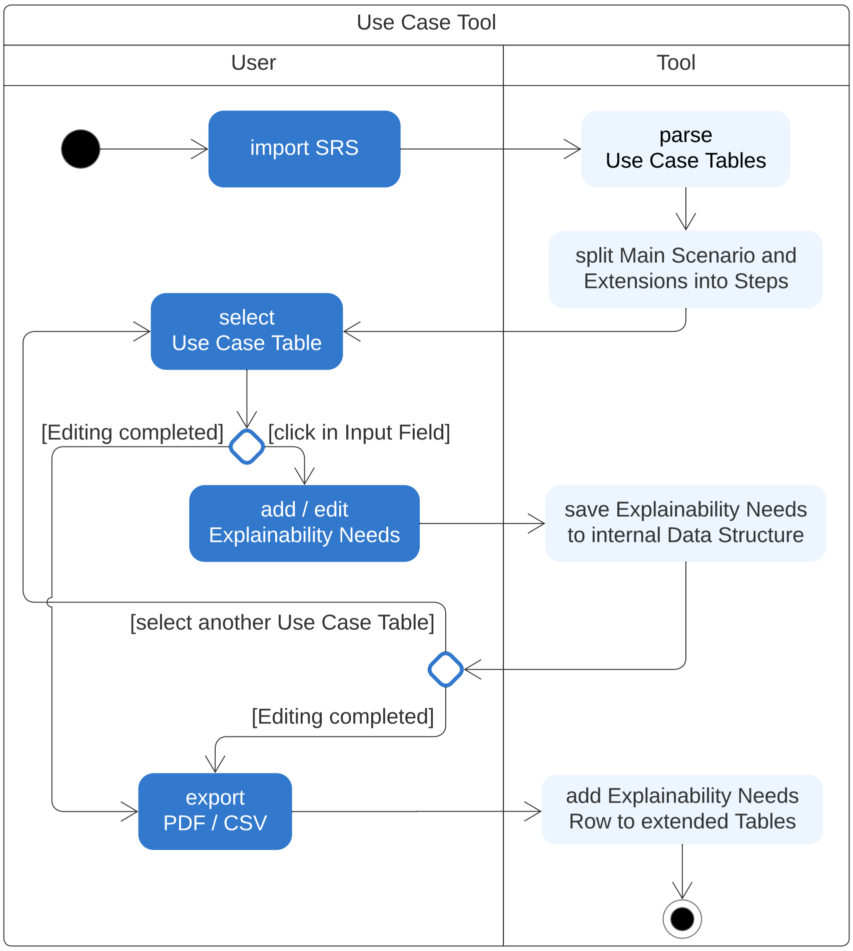
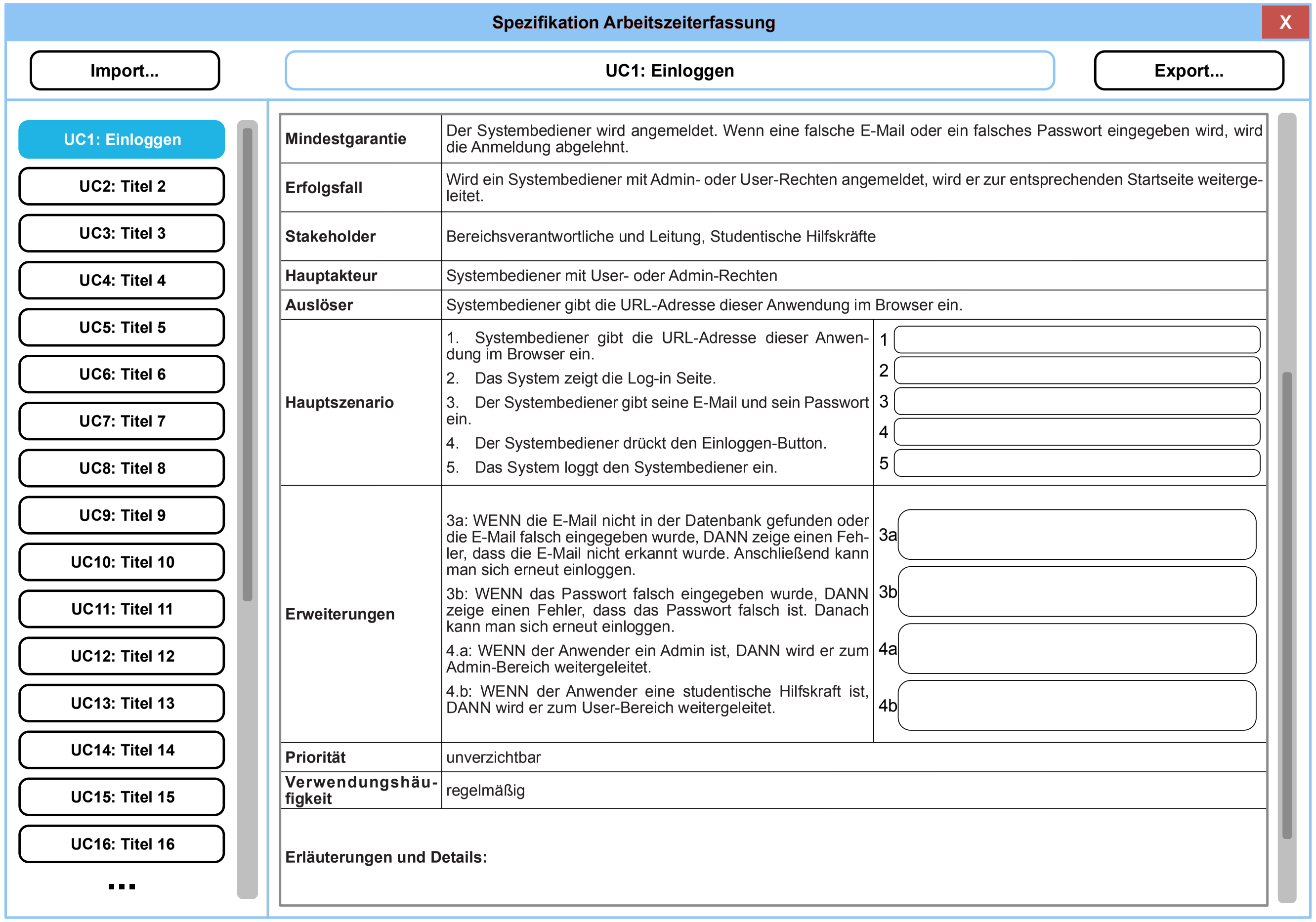


Figure 3.1: Activity diagram of the 'Use Case Tool'

3.5 MockUp

Based on the functional and non-functional requirements gathered from the thesis supervisor, the mockup shown in Figure 3.2 was created. It includes an import button to parse the Use Case tables of an SRS (R01) and an export button to export the Use Case tables with explainability needs (R04, R05). On the left are buttons to select the different Use Case tables (R02) and on the right are input fields for the user to add explainability needs to each main scenario and extensions step (R03). The language of the mockup is German to satisfy constraint NR03.



Mindestgarantie	Der Systembediener wird angemeldet. Wenn eine falsche E-Mail oder ein falsches Passwort eingegeben wird, wird die Anmeldung abgelehnt.	
Erfolgsfall	Wird ein Systembediener mit Admin- oder User-Rechten angemeldet, wird er zur entsprechenden Startseite weitergeleitet.	
Stakeholder	Bereichsverantwortliche und Leitung, Studentische Hilfskräfte	
Hauptakteur	Systembediener mit User- oder Admin-Rechten	
Auslöser	Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein.	
Hauptszenario	1. Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein. 2. Das System zeigt die Log-in Seite. 3. Der Systembediener gibt seine E-Mail und sein Passwort ein. 4. Der Systembediener drückt den Einloggen-Button. 5. Das System loggt den Systembediener ein.	1 2 3 4 5
Erweiterungen	3a: WENN die E-Mail nicht in der Datenbank gefunden oder die E-Mail falsch eingegeben wurde, DANN zeige einen Fehler, dass die E-Mail nicht erkannt wurde. Anschließend kann man sich erneut einloggen. 3b: WENN das Passwort falsch eingegeben wurde, DANN zeige einen Fehler, dass das Passwort falsch ist. Danach kann man sich erneut einloggen. 4.a: WENN der Anwender ein Admin ist, DANN wird er zum Admin-Bereich weitergeleitet. 4.b: WENN der Anwender eine studentische Hilfskraft ist, DANN wird er zum User-Bereich weitergeleitet.	3a 3b 4a 4b
Priorität	unverzichtbar	
Verwendungshäufigkeit	regelmäßig	
Erläuterungen und Details:		

Figure 3.2: UI Mockup of the ‘Use Case Tool’

3.6 Explainability Requirements

Throughout its development, the ‘Use Case Tool’ has been constantly reviewed and refined. During this process, the following explainability requirements have been raised. These are also prioritized from highest to lowest importance.

ER01 After starting the tool, a brief description of its features should be displayed.

ER02 An explanation of explainability needs should be included above the input fields.

ER03 The list of Use Case table titles should contain an icon when an input field of a Use Case table has been edited.

In order to inform users about the possibilities of the ‘Use Case Tool’, a brief description of its features should be displayed after starting it (ER01). This is the most important explainability requirement. The next important one is that an explanation of explainability needs should be included above the user input fields (ER02). This should be done to clarify what users are supposed to enter. The final explainability requirement is an icon that should appear in each table in the list of Use Case table titles when an input field has already been edited to give the user an overview on his or her edited Use Case tables (ER03).

Chapter 4

Implementation

This chapter covers the prerequisites, as well as the development process of the 'Use Case Tool' and the difficulties along the way. While the prerequisites include the choice of frameworks and libraries, the development process includes the main features and add-ons of the tool.

4.1 Prerequisites

At the beginning of the implementation, the question was which programming language to use and which frameworks and libraries would help to meet the requirements. The 'Use Case Tool' was to be developed preferably on macOS, and one of the requirements was that it should run on Windows 11. To ensure that the UI would look the same on both macOS and Windows during the development process, UI frameworks were researched and tested. The first framework considered was Uno platform [53], which supports all current operating systems and is open source. Once installed, all attempts to run a sample project failed due to various NuGet errors. NuGet manages libraries developed by others that can be used in .NET developments [40]. Afterwards, Avalonia (Section 2.6) was discovered and tested. The setup was flawless on both, macOS and Windows 11 and official sample projects provided looked identical on both platforms, making it a promising candidate. Since Avalonia is free for commercial use, platform independent, and therefore cost-effective and easy to update over the long term, it was the obvious choice. As Avalonia applications are written in a .NET language [6], the chosen programming language for the 'Use Case Tool' is C#. In addition, the MVVM pattern (Section 2.7) is used to keep development and maintainability more flexible.

The main purpose of the tool is to parse Use Case tables from an SRS in copyable PDF format and display them in the UI. To accomplish this, a library was needed to parse the table contents. This was necessary because each cell had to be read and then written to an internal data structure, which was not easily done with a text parsing library alone. During the research, Tabula (Section 2.8) stood out, especially for parsing table contents of copyable PDF files.

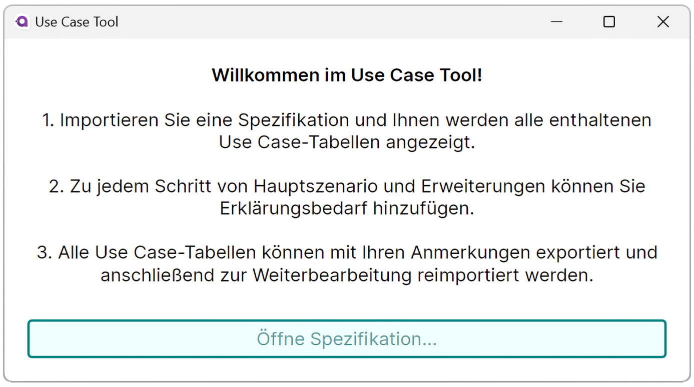
The MigraDocCore [47] library is used to enable the PDF export. It can generate documents by adding for example paragraphs and tables to sections. If tables do not fit on one page, they are automatically split into two pages based on the defined margins. Since it is open source under the MIT license just like Tabula, it is free to use and commercially usable.

4.2 Development Process

This section describes the development process of the 'Use Case Tool'. During the development, errors and difficulties were handled in an agile manner at all times.

4.2.1 User Interface

The first view a user sees when opening the 'Use Case Tool' is the `ImportView` shown in Figure 4.1. It contains a brief description of the available features and a button to import an SRS or a PDF previously exported with the tool. The button action is bound to the `OpenFilePicker` method (see Section 4.2.2).



Willkommen im Use Case Tool!

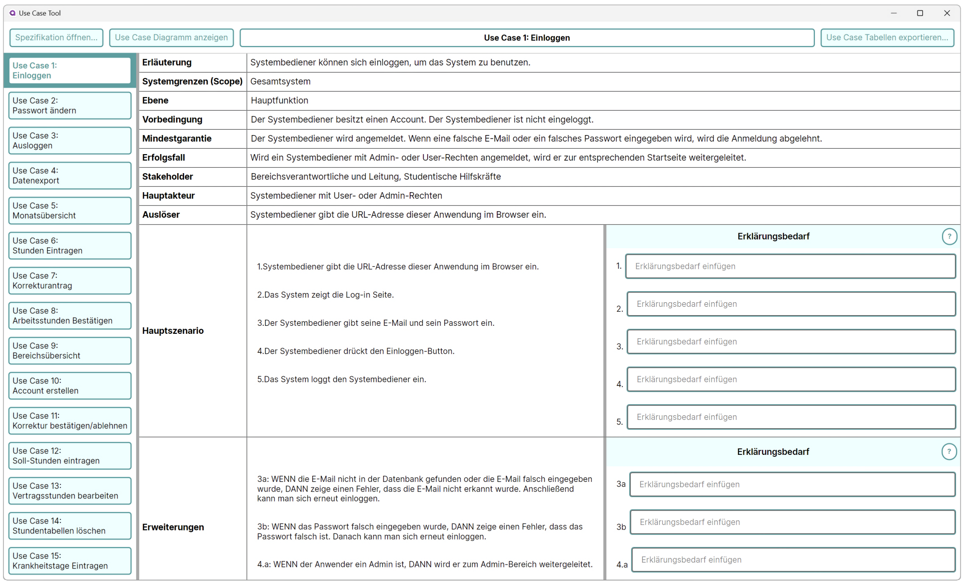
1. Importieren Sie eine Spezifikation und Ihnen werden alle enthaltenen Use Case-Tabellen angezeigt.

2. Zu jedem Schritt von Hauptszenario und Erweiterungen können Sie Erklärungsbedarf hinzufügen.

3. Alle Use Case-Tabellen können mit Ihren Anmerkungen exportiert und anschließend zur Weiterbearbeitung reimportiert werden.

Figure 4.1: ImportView of the 'Use Case Tool'

After importing, the `UseCaseView` shown in Figure 4.2 is displayed full screen. It consists of two rows in a `Grid`. One for the top bar with the import, Use Case diagram, and export buttons, and the second with the Use Case tables. The second row is itself a `Grid` with three columns. On the left, the first column displays a `ListBox` containing a dynamic number of Use Case table titles of an imported file, as the number varies for each SRS. To provide visual feedback to the user when selecting a Use Case table, the `Border` is implemented in two styles: one for unselected items and one for a selected item. This is controlled by the IsVisible property, which is bound to the IsSelected property of a ListBoxItem. The second column is for a GridSplitter, which allows to change the width of the ListBoxItems by the user. This can be useful if a title of the tables is very long, since the ListBox is oriented to the length of the title. The third column on the right displays the contents of a selected Use Case table in a DataGrid. This contains two DataGridTextColumns: one for the first column of a Use Case table and one for the second column and user input. The second column is implemented as a Grid with three columns. The first of these three columns is for the contents of the second column of the Use Case table. This column is bound to an internal data structure: either one string as Content and an empty ContentList or, in the case of the main scenario or extensions, an empty Content and multiple strings (steps) as ContentList. If the ContentList is empty, meaning that the current row is neither a main scenario nor extensions, the first column is stretched across all three columns. To achieve this, a Converter is used that adapts the emptiness of ContentList, or the inverse of it, to a Boolean value so that it can be bound to the IsVisible property of the controls. If the current row is a main scenario or extensions, the first of the three columns contains all steps in a ListBox. The second of the three columns is a GridSplitter, so that the width of the second and third column can be changed by dragging the bar between them. The third of the three columns is again a ListBox to display all the numbers from the main scenario or the extensions, followed by a TextBox to allow user input in the UI.


Use Case Tool
Spezifikation öffnen...
Use Case Diagramm anzeigen
Use Case 1: Einloggen
Use Case Tabellen exportieren...
Use Case 1: Einloggen
Use Case 2: Passwort ändern
Use Case 3: Ausloggen
Use Case 4: Datenexport
Use Case 5: Monatsübersicht
Use Case 6: Stunden Eintragen
Use Case 7: Korrekturantrag
Use Case 8: Arbeitsstunden Bestätigen
Use Case 9: Bereichsübersicht
Use Case 10: Account erstellen
Use Case 11: Korrektur bestätigen/ablehnen
Use Case 12: Soll-Stunden eintragen
Use Case 13: Vertragsstunden bearbeiten
Use Case 14: Stundentabellen löschen
Use Case 15: Krankheitstage Eintragen
?

Erläuterung	Systembediener können sich einloggen, um das System zu benutzen.	
Systemgrenzen (Scope)	Gesamtsystem	
Ebene	Hauptfunktion	
Vorbedingung	Der Systembediener besitzt einen Account. Der Systembediener ist nicht eingeloggt.	
Mindestgarantie	Der Systembediener wird angemeldet. Wenn eine falsche E-Mail oder ein falsches Passwort eingegeben wird, wird die Anmeldung abgelehnt.	
Erfolgsfall	Wird ein Systembediener mit Admin- oder User-Rechten angemeldet, wird er zur entsprechenden Startseite weitergeleitet.	
Stakeholder	Bereichsverantwortliche und Leitung, Studentische Hilfskräfte	
Hauptakteur	Systembediener mit User- oder Admin-Rechten	
Auslöser	Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein.	
Hauptszenario	1.Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein. 2.Das System zeigt die Log-in Seite. 3.Der Systembediener gibt seine E-Mail und sein Passwort ein. 4.Der Systembediener drückt den Einloggen-Button. 5.Das System loggt den Systembediener ein.	**Erklärungsbedarf** 1. Erklärungsbedarf einfügen 2. Erklärungsbedarf einfügen 3. Erklärungsbedarf einfügen 4. Erklärungsbedarf einfügen 5. Erklärungsbedarf einfügen
Erweiterungen	3a: WENN die E-Mail nicht in der Datenbank gefunden oder die E-Mail falsch eingegeben wurde, DANN zeige einen Fehler, dass die E-Mail nicht erkannt wurde. Anschließend kann man sich erneut einloggen. 3b: WENN das Passwort falsch eingegeben wurde, DANN zeige einen Fehler, dass das Passwort falsch ist. Danach kann man sich erneut einloggen. 4.a: WENN der Anwender ein Admin ist, DANN wird er zum Admin-Bereich weitergeleitet.	**Erklärungsbedarf** 3a Erklärungsbedarf einfügen 3b Erklärungsbedarf einfügen 4.a Erklärungsbedarf einfügen

Figure 4.2: UseCaseView of the ‘Use Case Tool’

4.2.2 Import

First, an `OpenFilePicker` method is implemented. To be able to import files with umlauts or spaces, the path parameter is decoded right away at the beginning of the function. an SRS in copyable PDF format can be imported, as well as Use Case tables previously exported with the tool as PDF. An iteration through all pages is implemented using Tabula. Tabula for C# has two different modes: the *stream mode* with the `BasicExtractionAlgorithm` and the *lattice mode* with the `SpreadsheetExtractionAlgorithm` [10]. The *lattice mode* is chosen because it parses each cell of the Use Case tables separately, making it easier to transfer the contents to the internal representation of the data structure. Then, an iteration through all the tables on a page found by Tabula is implemented with the constraint of two columns. This helps to skip all tables that cannot come into consideration for the Use Case table template that the Software Engineering Group is using. If a table with two columns is found, all contents are checked for parsing. To provide a basis for the following parsing steps, the different new line representations of the operating systems of a string in each cell are replaced by a uniform “\n”. The first check for a Use Case table is that the table starts with the string “Use Case”. Then, a new `UseCase` instance is created and the title is added. The first and second columns of a table are parsed together after checking the contents of the first column, which is described in the following. The content of the second column in each row can be added as a string to `Content` of the `UseCase` instance if the first column does not start with “Hauptszenario” (main scenario), “Erweiterung” (extension), or “Erklärungsbedarf”. If it does start with “Hauptszenario” or “Erweiterung”, the content has to be added as a dynamic list to the `ContentList` of the `UseCase` instance because there are different numbers of steps. Regular expressions are used to separate the steps. The regular expression “\n\d+[.:]?[a-z]?(?:[.:]\d+\s*)*” is used to separate multiple steps by a newline and a subsequent number, which can be followed by a period or colon, and a letter, which can also be followed by a period or colon. For example, “\n2.”, “\n2a:” “\n2.a”, “\n2:a:” are recognized patterns to split steps. The extraction of the first step is done with the same regular expression, but without the leading “\n”. A `SplitEnumeration` method is implemented to split the number (`Number`), the text behind the number(`Content`), and an input field (`InputText`) of each step, which are stored as a `NumberContentInput` property. If the content of the first column of a table starts with “Erklärungsbedarf”, this means that Use Case tables have already been edited and exported with the tool. This content is therefore reimported and allows users to edit previous input in the UI. To be able to edit the previous written explainability needs, each need from the `ContentList` according to main scenario and extensions has to be split and added to the `NumberContentInput` property. To achieve this, different regular expressions have to be used additionally with the SplitEnumeration method. The regular expressions are the same as above, but with a trailing “H:” for the main scenario and a trailing “E:” for the extensions (see Section 4.2.3). The steps of the “Erklärungsbedarf” ContentList have to be written into the correct InputText of the NumberContentInput property of main scenario or extensions. Two loops are implemented to iterate through the main scenario or extensions ContentList and “Erklärungsbedarf” ContentList to find the matching Number of the main scenario or extensions and the InputText from the “Erklärungsbedarf” ContentList. After all Use Case tables have been parsed, the contents can be displayed in the UseCaseView. A *Transitioning Content Control* [9] is chosen to be able to switch between different contents in a window. This helps to switch from the ImportView with the first import option to the UseCaseView and could be easily extended by more views in the future.

4.2.3 Export

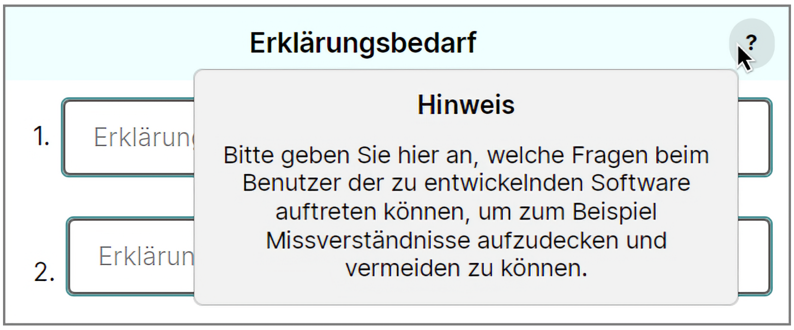
An ExportButton_Clicked method is implemented in the UseCaseView. Since PDF and CSV files have to be exported, two different ways had to be implemented. The PDF export uses MigraDocCore to create a new document and add a section and a paragraph. An iteration through all parsed and cached Use Case tables is implemented, adding a new table and the first row with the Use Case table title in two columns. This is followed by an iteration through all rows of a cached Use Case table. The content of the first column of the cached table is then added to the first column of the new row. If the first column contains the main scenario or the extensions, each Content of its ContentList is added to the allContents string, which is then added to the second column of the new row. In addition, each InputText of its ContentList that contains the explainability needs, is added to the allInputsMainScenario or allInputsExtension string with its Number. If it is not the main scenario or the extensions the second column can be added immediately to the second column of the new row, since it contains only a string. A Boolean is implemented to ensure that the explainability needs added in the UI are inserted in the right row. If the Boolean is true, they are added to the next row, which means below the main scenario row if no extensions are available, or below the extensions. In addition, an “H:” for ‘Hauptszenario’ is added to all numbers of InputText that belong to the main scenario, and an “E” for ‘Erweiterungen’ is added if they belong to the extensions. After iterating through all Use Case tables, a new instance of the *PdfDocumentRenderer* is created and the document is added and rendered into a PDF file. For the CSV export, a new object of the StringBuilder is instantiated. An iteration through all Use Case tables is implemented, which joins all columns in a row with a semicolon. A semicolon is used because commas are often used in written language, which can cause problems in the exported file. As with the PDF export, a Boolean is used to insert the explainability needs row in the right place, and the ContentList of main scenario and extensions is iterated through to assemble various strings with its Number and InputText. Each row of a Use Case table is appended separately to create the same number of rows as with the PDF export. The steps of the main scenario, extensions and explainability needs rows are concatenated with a semicolon so that each step has its own column cell. Finally, the StringBuilder instance is appended to a CSV file.

4.2.4 Add-Ons

This section presents all the add-ons that have been implemented in addition to the main features.

Explainability Needs Hint

During the implementation, it was discovered that interaction explanations for the input fields were missing. To achieve that users would know what to write in them, the outstanding text "Erklärungsbedarf" is written above the input fields which is supported by a hint "?" that has a hover effect with further explanation (see Figure 4.3). In addition, a placeholder was added to the input fields.



Hinweis

Bitte geben Sie hier an, welche Fragen beim Benutzer der zu entwickelnden Software auftreten können, um zum Beispiel Missverständnisse aufzudecken und vermeiden zu können.

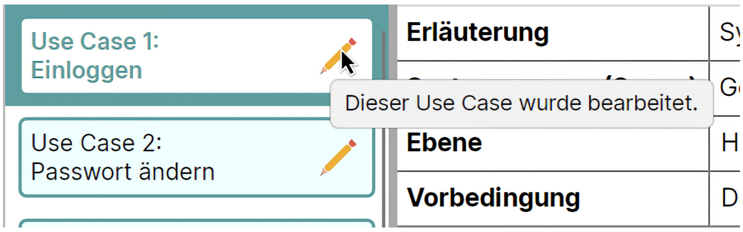
Figure 4.3: Hint for Explainability needs for the input fields of the 'Use Case Tool'

Use Case Diagram

The UseCaseDiagramWindow can be opened as a separate window using the button on the top of the UseCaseView. To access the diagram, all images are parsed in the OpenFilePicker method. They are cached in a list of IPdfImage, which is an interface of the PdfPig library [51] that is installed with Tabula. To be able to display an image in the UI, each image in the list is converted to a Bitmap, which is stored in a tuple with the page number it was found on. This list is passed to the `UseCaseDiagramViewModel`, which selects the image that is on the same page as the first Use Case table or on the previous page. If no image is found on these pages, or if the image is not in PNG or JPEG format, the window displays a note with this information.

Pencil Icon

Because users may want to review their previously added explainability needs in the UI before exporting, the pencil icon in the list of Use Case tables clearly indicates which Use Cases already have been supplemented with explainability needs. When something is written into an `InputText`, a Boolean is getting notified that is bound to each `ListBoxItem` on the left in the `UseCaseView`. The icon was self-designed, has a hover effect with an explanation, and is displayed on the left of a Use Case table button in the list, as shown in Figure 4.4.

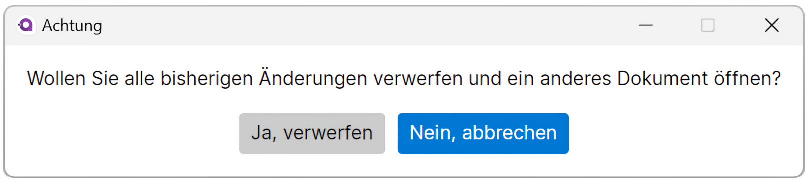


Dieser Use Case wurde bearbeitet.

Figure 4.4: The Pencil Icon in the list of Use Case table titles indicates that a table has been edited

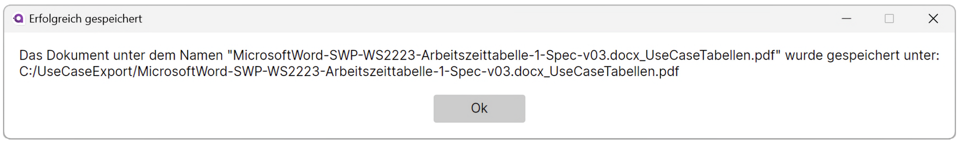
Message Boxes

Message boxes are implemented to prevent accidental closing and to provide feedback that the export was successful. The MessageBox.Avalonia [5] library is used, which allows the definition of custom buttons. The message boxes are used as a warning when the user clicks the import button in the `UseCaseView` or wants to close the whole tool (see Figure 4.5). After a file has been exported, a message box is used to indicate that the file has been successfully saved and to display the name and absolute path (see Figure 4.6).



Wollen Sie alle bisherigen Änderungen verwerfen und ein anderes Dokument öffnen?

Figure 4.5: Message box that warns the user when clicking the import or close button



Das Dokument unter dem Namen "MicrosoftWord-SWP-WS2223-Arbeitszeittabelle-1-Spec-v03.docx_UseCaseTabellen.pdf" wurde gespeichert unter:
C:/UseCaseExport/MicrosoftWord-SWP-WS2223-Arbeitszeittabelle-1-Spec-v03.docx_UseCaseTabellen.pdf

Figure 4.6: Message box indicating successful export

4.3 Difficulties

During the implementation of the import, many difficulties were encountered in parsing the Use Case tables as a whole. The first one was that the *stream mode* of Tabula was used where each cell was not parsed separately and the texts were mixed. This was solved by using the *lattice mode*. Another problem was that many tables were split on two pages somewhere between their rows or, in the worst case, between the steps of the main scenario or the extensions. This was first solved with a helper string and the prerequisite that the entire first Use Case table had to be written on one page. The last row string of the first column was written into the helper string, so that the following tables could be checked whether they were split or not. Since this was a big limitation, it was later changed to a helper Boolean that is true if a table with “Use Case” is found. This way a new `UseCase` object is instantiated only if the Boolean is false, and it is clear when the rest of a table is on another page and can be added to the last parsed table. In some sample SRSs, Tabula recognized the same Use Case table or some of its rows multiple times, or merged two different Use Case tables when they were on the same page. To avoid adding a second table, another helper Boolean is used to check if a Use Case table was found and the first row contains the string “Use Case” a second time. If the Boolean is true, all subsequent content is not parsed. A method `CheckForDuplicatedContent` is implemented to check if a row is already contained in the current Use Case table and if so, the row is skipped. Tabula can sometimes recognize the content of a split cell on the next page, and if it does, the text can be added to the previous row. Here it was important to distinguish between `Content` and `ContentList`. A parsed string can easily be added to the `Content`, while in a `ContentList`, its last step must be found and the parsed string must be split into `NumberContentInput` to add it as a step. SRSs are written by different people, which leads to the fact that the enumeration steps of the extensions are written differently. The problem is to define a generic but useful regular expression that, in the best case, recognizes all of them. In order to be able to reimport and further edit the `InputText`, the import and export to a PDF file had to be carefully implemented, which took a lot of testing time and readjustment.

Chapter 5

Study Design

After implementing the 'Use Case Tool', a user study was conducted. The aim was to test the usability of the tool as it is supposed to be used in future studies by the Software Engineering Group. This means that other people outside the institute will have to use the tool. Therefore, it should be easy to use and self-explanatory. The following chapter describes how the usability test was designed, conducted and executed. This also includes the selection of participants.

5.1 Participants

The participants selected for the usability test had to have a certain level of prior knowledge. This means that they must already have experience with the German SRS template defined by the Software Engineering Group of Leibniz University, in particular with the Use Case tables. The Use Case tables in this SRS template are based on the template by Cockburn [17]. There was no need for the attendees to have prior knowledge about explainability.

A total of nine participants took part in the usability test. Convenience sampling from personal and student networks was used to recruit the attendees. Accordingly, the participants are students of computer science at Leibniz University who are at the end of their Bachelor's degree or have already completed it. At this point, the participants should have acquired all the necessary skills to navigate and operate the system, and are therefore qualified to participate in a generalizable usability test [50].

5.2 Procedure

The setup for the study was a quiet room with tables, chairs and an external monitor that could be connected to a 14-inch laptop. Due to the NFR NR01 described in Section 3.3, the study is conducted on a laptop running a Windows 11 operating system with the latest version of the 'Use Case Tool' installed. An external monitor was chosen because NR02 (Section 3.3) defines that the tool should be usable on a screen of at least 16 inches in size. In addition, a computer mouse was connected to the laptop so that users would not be negatively affected by using the laptop's possibly unfamiliar trackpad.

The participants were invited to a room at the Software Engineering Group of the Leibniz University in early February 2024. A maximum time slot of 45 minutes was allocated for each attendee. At the beginning, each participant received a brief introduction to what the 'Use Case Tool' was developed for. This includes that the explainability needs should be identified as early as possible in the software development process and should be addressed after the SRS has been written. It was also mentioned that the usability test they are participating in does not aim to generate research results regarding explainability needs, but to test the usability and functionality of the tool. After the introduction, the participants tested the tool. Data was collected during and after the usability tests. The data collection process can be seen as a flowchart in Figure 5.1.

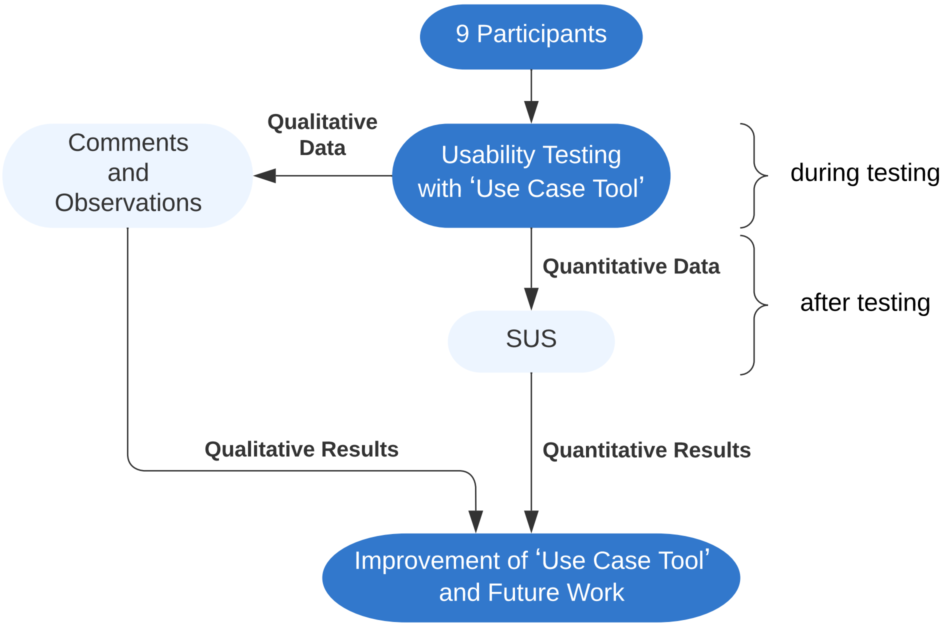


Figure 5.1: Data collection process of usability testing

5.3 Data Collection

Data was collected both quantitatively and qualitatively following the approach of Pruitt and Adlin [44]. Typically, quantitative data is collected through questionnaires or surveys, while qualitative data is collected through interviews or workshops. In addition, a large number of people tend to be surveyed quantitatively, while a smaller number of people tend to be surveyed qualitatively. The collection of all data was completely anonymous.

Qualitative data was collected while the participants tested the usability of the tool. Each participant was asked to verbalize their thoughts during the use of the tool according to the think aloud method. In this way, positive and negative aspects could be noted and used to improve the tool in the future [33]. In addition, the attendees were observed during use to capture spontaneous, uncommented reactions. If the participants felt that they had tested everything, but certain main features had not yet been tested, they were made aware of this.

Quantitative data was collected after the 'Use Case Tool' was tested. The participants were asked to fill out a questionnaire according to Brooke's SUS assessment [12]. The original SUS statements shown in Table 2.3 have been adapted, but only slightly modified (cf. Table 5.1), to refer to the developed tool. As the participants are German, the questionnaire was translated into German. A 5-point Likert scale ranging from 1 - 'Strongly Disagree' to 5 - 'Strongly Agree', shown in Figure 2.2, was used to determine each participant's level of agreement. An overview of the SUS questionnaire and the Likert scale used in the study can be found in Appendix B.

5.4 Data Analysis

After usability testing was finished, the collected data was analyzed. The quantitative data from the 5-point Likert scale SUS was used to calculate the SUS score for each participant. The calculation of the score is described at the end of Section 2.4. The qualitative data, the notes collected during usability testing, were sorted, viewed and evaluated alone.

Number	**Statement**
1	I can well imagine using the tool regularly in a work context.
2	I think the tool is unnecessarily complex.
3	I find the tool easy to use.
4	I think I would need technical support to use the tool.
5	I find that the various functions of the tool are well integrated.
6	I think there are too many inconsistencies in the tool.
7	I can imagine that most people who work with software projects and specifications will quickly become familiar with how to use the tool.
8	I find the tool very cumbersome to use.
9	I felt very confident using the tool.
10	I had to learn a lot of things before I could work with the tool.

Table 5.1: Modified usability test statements for the ‘Use Case Tool’

Chapter 6

Results

This chapter describes the results of the usability testing of the ‘Use Case Tool’. It is divided into quantitative and qualitative data to be evaluated.

6.1 Quantitative Results

The data collected with the statements in Table 5.1 according to the SUS is shown in Figure 6.1 as a diverging stacked bar chart. This figure also shows that the statements of a SUS are alternately positive and negative. The odd-numbered statements are most often rated as ‘Agree’ and ‘Strongly Agree’, and the even-numbered statements are most often rated as ‘Disagree’ and ‘Strongly Disagree’.

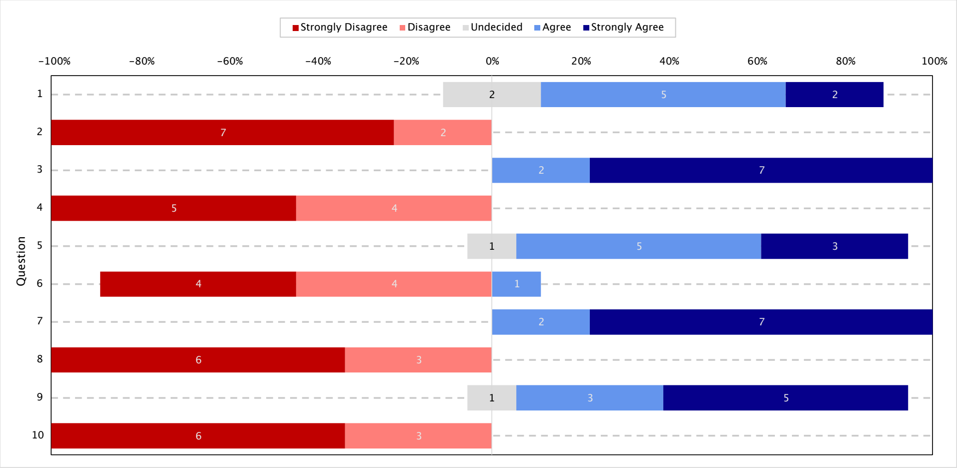


Figure 6.1: Usability test answers of all participants

All nine participants found the ‘Use Case Tool’ easy to use (statement 3) and could imagine that stakeholders would quickly become familiar with its use (statement 7). Seven out of nine attendees (77.8%) could imagine using the tool regularly in a work context (statement 1).

No one found the 'Use Case Tool' to be unnecessarily complex (statement 2) or cumbersome (statement 8). This is also reflected in the statements 4 and 10, where no one thought that technical support would be necessary to use the 'Use Case Tool' (statement 4) or that they had to learn a lot to be able to work with the tool (statement 10).

Eight participants (88.9%) felt that the 'Use Case Tool's' features were well integrated (statement 5) and that the tool had no inconsistencies (statement 6). One respondent (11.1%) felt that there were inconsistencies in the Use Case Tool (statement 6). Eight out of nine attendees (88.9%) felt very confident using the 'Use Case Tool' (statement 9). Figure 6.2 shows the calculated SUS score for each participant.

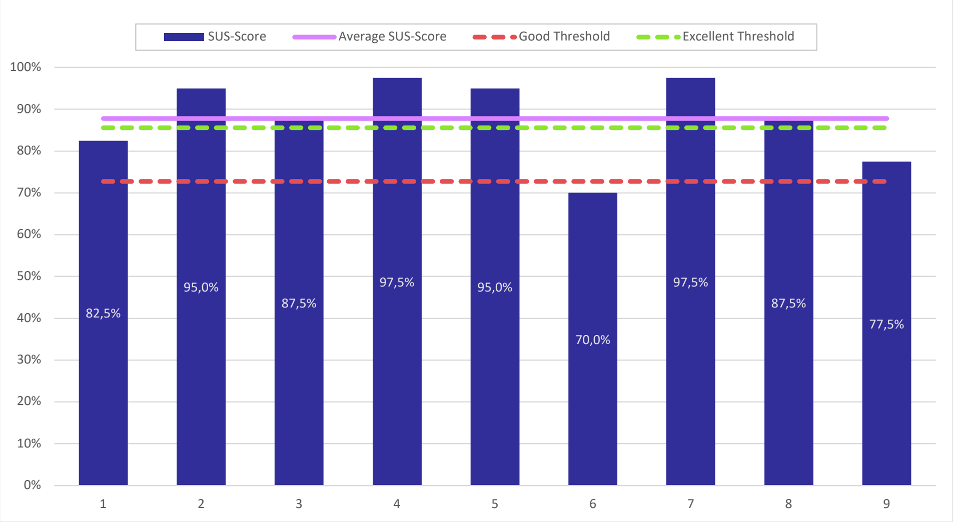


Figure 6.2: SUS score for each participant and the average

The thresholds drawn in Figure 6.2 were taken from Bangor et al. [1], shown in Figure 2.2. Eight scores are above the threshold for good usability (72.8%), one is slightly below good usability, and six are above the threshold for excellent usability (85.6%). The average SUS score is 87.8%, which is above the threshold for excellent usability.

6.2 Qualitative Results

Table 6.1 shows *pattern codes* [45] for the comments made by the participants and observations made by the experimenter during the usability tests in the second row. The first row shows the number of participants who mentioned the text in the second row, reacted to a feature or forgot to test a feature.

Three participants (33.3%) tried using the tabulator key to move to the next input field. After this did not work, they commented that this would

# People	Comments and Observations
9	Irritated by export message box
9	Had to think about “H” and “E” in exported Use Case tables
9	Did not try export as CSV file by themselves
9	Tried CSV import after export was possible
8	Wanted to zoom in on Use Case diagram
5	Needed hint for reimport
4	Would write out “H” and “E” and place them elsewhere
3	Wanted to use tabulator key for input fields
3	Searched a save button
2	“Spezifikation öffnen...” misleading
2	“Spezifikation öffnen...” message box irritating if nothing changed
2	Clicked on hint ‘?’
2	Found diagram display error
2	Said that Pencil Icon is helpful
1	Confused why diagram was not exported
1	‘Erläuterungen und Details’ was not shown in the UI
1	Export message box can not be closed with ESC
1	Confused about missing hint that CSV file cannot be reimported
1	UseCaseDiagramWindow in foreground is good
1	minWidth and minHeight not set on UseCaseView window
1	Diagram vanishes after “Spezifikation öffnen...”, “Ja, verwerfen” and “Abbrechen” clicked
1	Text after a written ‘;’ is in a new cell in exported CSV file

Table 6.1: Comments and observations from usability testing

be more convenient than working with the mouse.

All participants (100%) were visibly irritated by the message box containing a long path name that appears after clicking the save button in the export file picker. One participant (11.1%) tried to close this message box with the escape key and was frustrated that it did not work. Also, no one (100%) thought of exporting anything other than PDF files, so the experimenter gave a hint to look for other formats to export. When the attendees found out that they could export a CSV file, they all (100%) tried to import it as well. One participant (11.1%) asked why there is no hint that an exported CSV file cannot be reimported. Another one (11.1%) found out that text after a ‘;’ in the input text box is written in a new cell in the exported CSV file.

In the exported Use Case tables, all participants (100%) had to think about the meaning of the abbreviations “H” and “E” in the new row called explainability needs. Four attendees (44.4%) mentioned that they would write them out as “Hauptszenario” and “Erweiterungen” and that there could be a different row for each in the exported tables.

Five participants (55.6%) did not try to reimport an exported PDF file. They were given a hint by the examiner after they said that they do not know what else they could try. Two of the attendees (22.2%) said that the "Spezifikation öffnen..." ('Open specification...') button is misleading because the exported file contains only the Use Case tables and is not an SRS. The message box that appears when the "Spezifikation öffnen..." button is clicked was mentioned as confusing when nothing was edited by two participants (22.2%).

Eight of the participants (88.9%) tried to zoom in on the Use Case diagram. Two of them (22.2%) found a display error that appeared in different window sizes when the window was resized. One attendee (11.1%) was confused that the diagram was not exported to the PDF file. Another one (11.1%) found out that the diagram disappeared after clicking the combination of "Spezifikation öffnen...", "Ja, verwerfen" ('Yes, discard') and "Abbrechen" ('Cancel'). One participant (11.1%) was particularly positive about the fact that the `UseCaseDiagramWindow` stays in the foreground after opening.

Three attendees (33.3%) looked for a button to save the data during the test. Two (22.2%) clicked multiple times on the '?' that has a hover effect and contains a hint. The Pencil-Icon was described as helpful by two participants (22.2%). One attendee (11.1%) has noticed that the text "Erläuterungen und Details" ('Explanations and Details') under a Use Case table in the SRS is not available in the UI or in the exported files. It was mentioned that "Erläuterungen und Details" could solve some explainability needs, so it would be useful to include it. Another participant (11.1%) tried to resize the `UseCaseView` window and found that no minimum height or width is set, so the window can almost disappear completely.

Chapter 7

Discussion

This chapter discusses the results of usability testing, as well as the limitations and challenges that should be considered.

7.1 Discussion of Results

The average SUS score of 87.8% achieved during usability testing shows that the 'Use Case Tool' is above the threshold of an excellent SUS score of 85.6%, as defined by Bangor et al. [1]. It also indicates a high level of user acceptance (cf. Figure 2.2). This leads to the assumption that users should be able to use the tool easily. When looking at each SUS score individually, the results vary between users. A reason could be that each user has different expectations of the tool itself or some of its features. Only one user had a score of 70%, which is below the good threshold of 72.8%. The expectations of this user were not sufficiently met, but the acceptance is still in a high range according to Bangor et al. [1].

The qualitative results provide more insight into why users rated the 'Use Case Tool' as they did and may also reveal their expectations or problems in using the tool. All nine participants were confused by the message box that opens after clicking the save button on the export file picker. It should be noted that the message box contained a very long path name and a very long suggested name, which was derived from the actual file name. It was also noticeable that all participants accepted the proposed name. The purpose of this message box was to give the user feedback that the file had been saved, and also to display the path where the file was saved and its name. To minimize the confusion for the participants, the different texts could be displayed with a little more space between them for a better overview. Another possibility could be to remove the path and name and just display a short success message. One attendee tried to close the message box with the escape key. Since no one else tried this, it could be considered a minor feature, but it might be convenient not to have to switch back to the mouse. In addition, three participants tried to use the tabulator key to switch between the input fields because they also did not want to switch back to the mouse. This suggests that power users like the convenience of using only the keyboard.

After exporting the extended Use Case tables, all participants paused for a moment and had to think about the abbreviations “H” and “E” after the step number in the new row “Erklärungsbedarf”. Four of the attendees mentioned that they would prefer no abbreviations and one row of explainability needs for the main scenario and one for the extensions. It would be easy to write “Hauptszenario” and “Erweiterungen” instead of the abbreviations, but this could make the text in the row of explainability needs very long and confusing. The option with two different rows could also be implemented. However, this would require much more effort to modify the current implementation for the reimport.

None of the participants tried to change the format of the exported file. This suggests that they did not see the need to export to a format other than PDF, which was the default. The CSV format for export was chosen because it facilitates analysis when the tool is used to generate explainability needs studies. Therefore, it can be assumed that the direct users may not have the purpose to export a CSV file, but that the evaluators can derive a high benefit from it. One attendee found an error in the CSV export. When a ‘;’ is written in an input field, the text behind it is written in a new cell in the exported CSV file as if it belonged to a different step. This error needs to be fixed by banning the semicolon from the input field or replacing it with another punctuation mark. While reconstructing the error, it was discovered that a CSV file that was to be replaced by a new file with the same name simply added the new export to the old file. So there should be a warning that the data should not be saved over an existing CSV file.

It was interesting to see that all nine participants thought they could import a CSV file after being made aware of the CSV file export. They probably thought that if they missed something on the export file picker, they might have also missed something on the import file picker. This raises the question of whether they really wanted the CSV file import or just wanted to make sure they were not missing anything. One potential issue that came up was that one attendee was expecting a hint that a CSV file could not be reimported. This could lead to confusion and misunderstanding if someone exports a CSV file and tries to reimport it. Since this will not work, the user will not be able to continue working on these Use Case tables in the ‘Use Case Tool’. So there should be a note that CSV files cannot be reimported, or the reimport feature should be implemented.

Less than half of the participants (four out of nine) tried to reimport PDF files on their own. This was a bit unexpected, especially since the `ImportView` includes a message about the option to reimport files. Two attendees mentioned that the “Spezifikation öffnen...” button is misleading because the exported Use Case tables are no longer an SRS. This is an indication that the button text was poorly chosen or not well thought out. The button text is easily changeable and should be changed to something more generic like 'Open...' or 'Import...'.

The message box that opens when clicking the "Spezifikation öffnen..." button irritated two participants because they had not changed anything in the input fields and wanted to select a different file. This message box is intended to help users to not lose their progress when adding or editing the explainability needs of Use Case tables in an SRS. It serves as a warning that any added input will be lost if the Use Case tables have not been exported beforehand. During the usability tests the attendees focused on testing the tool's features and opened several files in a short time. This will probably not be the case when the 'Use Case Tool' is used for its intended purpose. So the question is whether making new changes dependent on the message box is a very important change.

Eight out of nine participants tried to zoom in on the Use Case diagram, showing that everyone expects to be able to zoom in on an image. However, since the Use Case diagram was seen as an add-on to the implementation, this was not the primary focus. Two attendees found a display error when resizing the `UseCaseDiagramWindow`. This is probably an error that can only be fixed with a different image extraction library. Another participant found an error which can and should be fixed: The diagram disappears after clicking the combination of "Spezifikation öffnen...", "Ja, verwerfen" and "Abbrechen". One attendee mentioned that it is a good choice to keep the `UseCaseDiagramWindow` in the foreground. That way, you can see the diagram and still write in the input fields. One participant was confused that the Use Case diagram did not export to the PDF file. This was tried, but unfortunately did not work, and since it is an add-on, it was not pursued further due to time constraints.

Three participants were looking for a button to save the contents of their edited Use Case tables. Since the export works like a 'save as', this was not considered. For the future it might be useful to implement an additional save button, but the question is whether this is necessary. During the usability tests, attendees switched between files a lot and did not read each Use Case table to add explainability needs, which the 'Use Case Tool' is designed to do.

The hover effect hint marked with a '?' above the input fields seems to be misleading for some people. Two of the participants clicked on it several times, while others did not. The design of the '?' could be changed. The question is what to change so that it is not misleading for some people.

One attendee discovered that the `UseCaseView` window did not have a minimum width and height set. The implementation contained an error, so the minimum width and height were not applied correctly. In general, the window size was intentionally not hard coded, allowing users to resize the window to their needs. However, a minimum width and height should definitely be set so that all buttons and contents are fully visible and usable. This is also required by NR02 (Section 3.3), which defines a window with a size of at least 16 inches to avoid truncated UI elements.

In an SRS, there can be text called 'Erläuterungen und Details' under each Use Case table. Currently, this cannot be displayed in the 'Use Case Tool', and one attendee mentioned that parsing this too could avoid some upcoming explainability needs. This is a future work because these texts are not a part of the table and Tabula (Section 2.8) cannot parse texts that are not in tables. So another library has to be found to parse these texts.

Two participants were particularly positive about the pencil icon. This shows that it can be a good explainability requirement to help find the Use Cases where something has already been edited.

7.2 Improvements

During usability testing, it was discovered that the minimum width and height in the `UseCaseView` window were not set correctly. This has been adjusted in both the `ImportView` and the `UseCaseView` window. The minimum size is set to a little under 16 inches, without truncating any UI elements. This is to avoid user restrictions that could cause problems on smaller laptops and user dissatisfaction. In this context, a minimum width has also been added for columns separated by a `Gridsplitter`.

Another problem discovered was that a semicolon in an input field would cause a new cell in the exported CSV file to start with the text after the semicolon. Therefore, the semicolon is now replaced with a '-' in the CSV export. The semicolon remains in the PDF export because it does not cause any problems. Since all participants were looking for the CSV file import after knowing about the export, and one participant asked for a note that the CSV reimport is not possible, the CSV reimport was implemented as well. While testing the CSV import, another error was found in the export, where the steps of the main scenario were again written in extensions. This has been improved by resetting a string.

The last error found was that the Use Case diagram was deleted after opening the import file picker by clicking "Spezifikation öffnen...", "Ja, verwerfen" and "Abbrechen". This was caught with an if clause that checks if the file picker was aborted, and some initial variables had to be moved after the if clause.

The "Spezifikation öffnen..." buttons were renamed to "Importiere..." ('Import...') after the usability tests (see Figures C.1 and C.3). The previous name was misleading and five participants did not try to reimport the exported files. 'Import' seems to be more appropriate than 'Open', because in the case of an SRS, the whole document is not opened, but only the Use

Case tables are imported. The text of the `ImportView` window has also been changed (see Figure C.1). There is now a fourth step describing that it is possible to reimport a PDF and CSV file. This should make users more aware of this option and prevent them from overlooking it as easily as in the previous text.

The message box with a lot of text that appeared after saving a file was irritating to all participants. To avoid this, the text has been shortened (see Figure C.2). The file name is included in the absolute path name, so only the absolute path name is displayed in the message box. It has also been decided to no longer use the suggested file name in the export file picker anymore, because in the case of Microsoft Word, the file name is always preceded by "Microsoft Word". This could also have caused confusion and a long message box text, as most users did not change the suggested file name. One participant suggested adding the ability to close this message box with the escape key. Since this already worked for another message box and was easy to implement, it was added here.

7.3 Limitation and Challenges

Since the study participants were obtained through convenience sampling from personal and student networks, there are some limitations to the validity of the results. The attendees may have *demand characteristics* [42] in favor of the 'Use Case Tool' adopted, which may have influenced their feedback to avoid being overly critical. *Demand characteristics* can occur for example, because participants know what an experiment is for and that they, consciously or unconsciously, want the result to be a certain way. This can influence the results of an experiment when working with humans [42]. Attempts were made to prevent *demand characteristics.* Before and during the usability test, attendees were told and encouraged to test anything that came to their mind, and that all notes taken by the experimenter would be anonymous. In addition, the anonymity of the responses was emphasized before the questionnaire was administered, and the experimenter did not look at the screen during the completion of the questionnaire until it was submitted. All participants were computer science students, which means that people with other backgrounds may have different problems using the tool than those already identified.

Another limitation is that the participants of the usability testing are not the secondary stakeholders for whom the tool was designed. They only tested the usability of the 'Use Case Tool' to find errors and evaluate its features. This does not correspond to the actual use to extend Use Case tables with explainability needs. Thus, the feedback given lacks the purpose of real use, which should be tested in another study with participants who have knowledge about explainability needs.

The Use Case tables of an SRS that can be imported into the 'Use Case Tool' require a specific layout. The tables must have two columns in order to parse the contents. In addition, the tables must contain the string "Use Case" in the first cell, as this is the marker for a Use Case table in the implementation. The Use Case tables also have to contain the string "Hauptszenario" for the main scenario and "Erweiterung" for the extensions. All of these hard coded strings limit the number of Use Case tables that can be parsed. The layout of the files that can be reimported is also restricted.

The regular expressions used to divide the steps of the main scenario and the extensions are generic, but especially the extensions steps need a specific form to be recognized. This form has to be a number and a period/colon, a number with a letter that may be followed by a period/colon or a number with a period/colon and a letter that may be followed by a period/colon. It is important that there is no space between the number, letter, and period or colon, otherwise the part after the space will not be recognized as a coherent step.

The implementation of the export feature has the limitation that it needs to have a row after the main scenario. This is the case because sometimes the Use Case tables contain extensions and sometimes they do not. In order to be able to add the explainability needs in the right row, there is one look ahead row to check if there are extensions after the main scenario in the table. If a table does not contain a row after the main scenario, the tool will crash when exporting.

The Tabula library sometimes has problems with text recognition. This can be seen in some SRS when sentences have no spaces, even though spaces are visible in the PDF. Recognition also sometimes fails when a table cell is split across two pages. To avoid this, another library could be tested. The text 'Erläuterungen und Details', which can be written below Use Case tables, is also not parsed because it is not written in a table and therefore cannot be parsed with Tabula. In order to parse this text, another text recognition library needs to be found.

A CSV file cannot simply be replaced with a new one during export. When a file is replaced, the new content is written below the content of the first export. This may cause problems and is a limitation, but it is not something that can be changed by the tool.

The Use Case diagram is currently determined by the page number it is on, so the image on the previous page or on the page of the first Use Case table is selected, which is unsafe. This could be improved by using a text recognition library to recognize the title "Use Case Diagramm" (use case diagram) and parse the image below. This is not possible with Tabula because it only checks for text in tables.

Chapter 8

Conclusion and Future Work

This chapter summarizes the goals of this thesis and the results of the usability testing of the developed 'Use Case Tool'. Furthermore, future work on the tool is discussed.

8.1 Conclusion

Identifying explainability needs early on in the development process of new software is a challenge, as its absence can lead to inefficiencies and limitations in the usability and comprehensibility of the software. In order to address this problem, the 'Use Case Tool' was developed for the Software Engineering Group of Leibniz University.

The first step was to elicit requirements from the supervisor of this thesis. To ensure that all requirements and ideas were met, a mock-up and activity diagram were created and reviewed. The requirements were prioritized, so that the development process was clearly structured. Throughout the development process, difficulties and constant feedback via weekly meetings were handled in an agile manner. This helped to further improve the tool's accuracy, comprehensibility, and capabilities. In addition, the tool is designed to be as self-explanatory as possible, so that it can be easily used for future studies by the Software Engineering Group.

The developed tool is designed to be used in the planning phase of new software. It is able to parse Use Case tables of an SRS and display them in the UI. Users can extend the main scenario and extensions steps with explainability needs that arise when reading a step. All parsed and extended Use Case tables can be exported to PDF and CSV format. To resume work after a break, both formats can be reimported.

Usability tests with nine participants were conducted to test the usability of the 'Use Case Tool'. The average score resulting from the SUS is 87.8%, which indicates excellent usability. The additional qualitative data collected provided more details about inconsistencies and errors, as well as positive elements of the tool. Overall, the study found that the main and additional features were well integrated. The identified errors and misleading text elements in the UI have been improved to make the tool even easier to understand and use. In conclusion, judging by the study and the further improvements, the 'Use Case Tool' seems promising to be used in future studies for the extension of Use Case tables with explainability needs and can support requirements engineering for explainability requirements in early stages of software development.

8.2 Future Work

In the study, the 'Use Case Tool' already achieved an excellent average SUS score. To further improve the tool, according to the results of the study, some features can be modified and added.

For example, the message box that opens each time the import button is clicked could only be displayed when an input field is newly edited. This way, users would not be confused as to whether or not they had edited an input field again. To switch between the input fields without using the mouse, the study revealed that some participants would prefer to use the tabulator key. This is another feature that can be considered.

The Use Case diagram is not currently exported to the PDF file, which may also be a feature to implement. Since this was mentioned in the study and the Use Case diagram shows an overview of all Use Case tables, future users may miss the diagram in the exported file. The same applies to 'Erläuterungen und Details', the text below the Use Case tables. This was also missed by one participant, and may contain explanations. Therefore, it is important to parse and add 'Erläuterungen und Details' to the 'Use Case Tool', as this may already clarify explainability needs that arise without it. To achieve this, another text recognition library must be used. A pure text recognition library could also help to make the parsing of the Use Case diagram more safe, because it can be searched for the title "Use Case Diagramm" and the image below can be parsed.

Import is currently only available for PDF files and reimport for CSV and PDF files. To extend the import for an SRS, LaTeX and Word files could be considered. Since LaTeX and Word files can be saved as copyable PDF files, the import is an additional feature for these files, as without it only one additional step is required when saving.

Since some participants in the study were looking for a save button, this could be added or the export button could be revised. One way to revise the export button is to add a menu bar at the top with the usual 'Save' and 'Save as...' options. In this case, the 'Ctrl' and 'S' key combination could also be implemented for power users. Since the words 'Save' and 'Save as...' can be misleading in the context of the 'Use Case Tool', they could be replaced with ‘Export’ and ‘Export as...’. If a menu bar is added, the import button could also be placed in it. Another option might be to save changes automatically, for example when selecting a different Use Case table or constantly when editing an input field. In this case, it needs to be defined where to save it and when to ask for export.

To make the ‘Use Case Tool’ more modular for different two-column tables, an option could be implemented to let the user define the hard coded strings as “Use Case”, “Hauptszenario”, “Erweiterung” and “Erklärungsbedarf”. This would allow for different Use Case table templates and other languages. More modularity could also be achieved by using Optical Character Recognition (OCR) to import a non-copyable PDF file.

Appendix A

Acceptance Testing

A.1 Test Case 1 - File Import

Setup: The program was installed and opened on the user's computer/laptop with the Windows 11 operating system.

Input	**Output**
The user presses the "Import..." button.	The program displays a file selection dialog.
The user selects the file "Working-time-table.pdf" in the dialog and then presses the "Öffnen" button.	The program opens the Use Case overview with a selection of the individual Use Cases on the left-hand side of the user interface and the content of the selected first Use Case "UC1: Einloggen" on the right.

A.2 Test Case 2 - Choose Use Case

Setup: The program was installed on the user's computer/laptop with the Windows 11 operating system, opened and the specification "Working-time-table.pdf" was imported.

Input	**Output**
The user selects "UC2: Passwort ändern" in the left-hand menu.	The program displays the content of "UC2: Passwort ändern" on the right-hand side.

A.3 Test Case 3 - Enrich Use Case with Explainability

Setup: The program was installed on the user's computer/laptop with the Windows 11 operating system, opened and the specification "Working-time-table.pdf" was imported. The user is in "UC2: Passwort ändern" in the user interface.

Input	**Output**
The user clicks on the text field for step 6 in the main scenario.	The program allows you to make an entry in the text field for step 6 of the main scenario.
The user inserts the text "Was passiert mit meinem alten Passwort? Wird es überschrieben?" into the text field.	The program displays the text "Was passiert mit meinem alten Passwort? Wird es überschrieben?" in the text field.

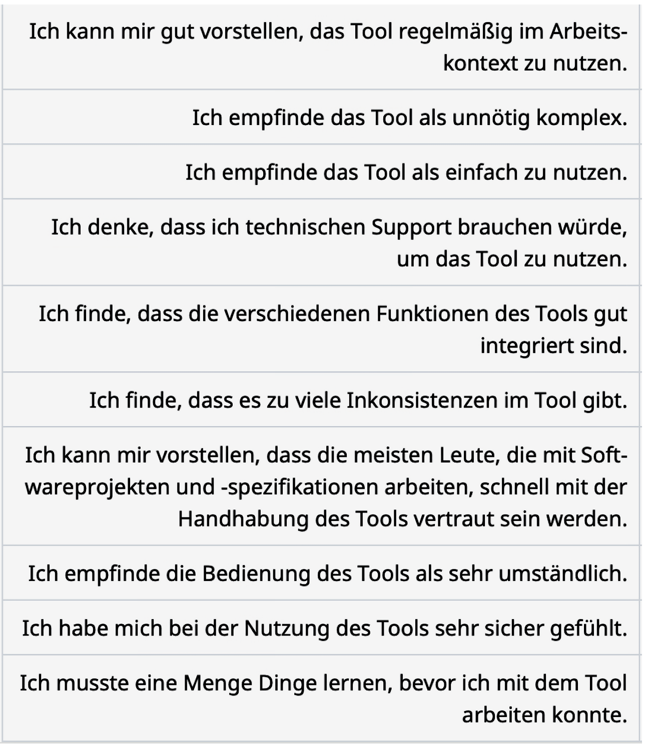
A.4 Test Case 4 - File Export

Setup: The program was installed on the user's computer/laptop with the Windows 11 operating system, opened and the specification "Working-time-table.pdf" was imported. The user has added a need for explanation in "UC2: Passwort ändern".

Input	**Output**
The user presses the "Export..." button.	The program opens a selection with PDF and CSV.
The user selects PDF format.	The program saves all Use Cases and the additional explanation in Use Case 2 in its table with a new row "Erklärungsbedarf" as a PDF file.

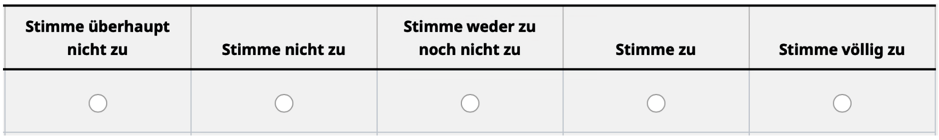
Appendix B

Usability Questionnaire



Ich kann mir gut vorstellen, das Tool regelmäßig im Arbeitskontext zu nutzen.
Ich empfinde das Tool als unnötig komplex.
Ich empfinde das Tool als einfach zu nutzen.
Ich denke, dass ich technischen Support brauchen würde, um das Tool zu nutzen.
Ich finde, dass die verschiedenen Funktionen des Tools gut integriert sind.
Ich finde, dass es zu viele Inkonsistenzen im Tool gibt.
Ich kann mir vorstellen, dass die meisten Leute, die mit Softwareprojekten und -spezifikationen arbeiten, schnell mit der Handhabung des Tools vertraut sein werden.
Ich empfinde die Bedienung des Tools als sehr umständlich.
Ich habe mich bei der Nutzung des Tools sehr sicher gefühlt.
Ich musste eine Menge Dinge lernen, bevor ich mit dem Tool arbeiten konnte.

Figure B.1: SUS questionnaire in German used in the study

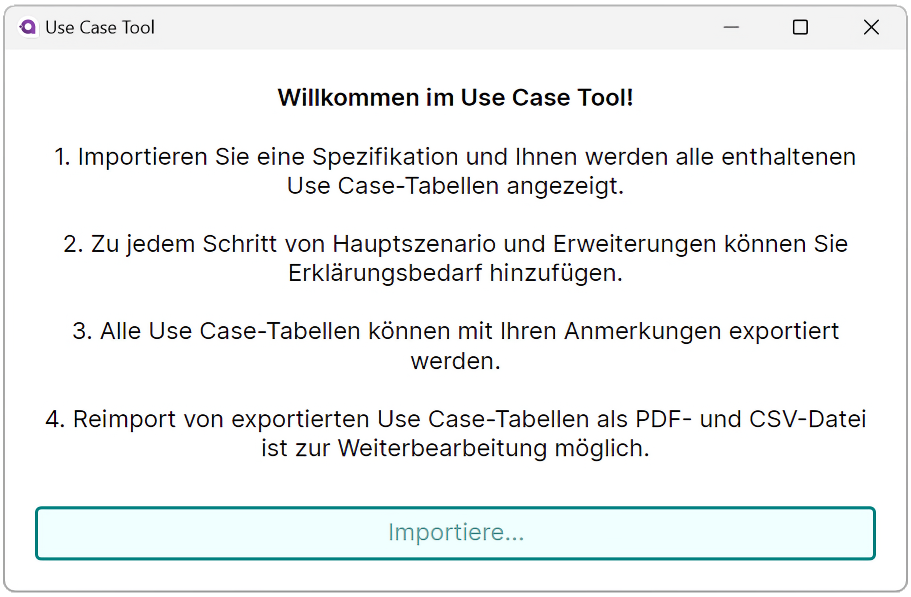


Stimme überhaupt nicht zu	Stimme nicht zu	Stimme weder zu noch nicht zu	Stimme zu	Stimme völlig zu
○	○	○	○	○

Figure B.2: Likert scale in German used in the study

Appendix C

Improvements Screenshots



Willkommen im Use Case Tool!

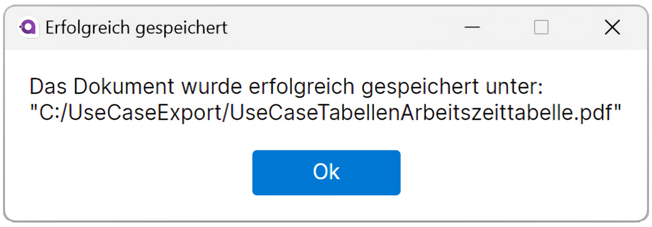
1. Importieren Sie eine Spezifikation und Ihnen werden alle enthaltenen Use Case-Tabellen angezeigt.

2. Zu jedem Schritt von Hauptszenario und Erweiterungen können Sie Erklärungsbedarf hinzufügen.

3. Alle Use Case-Tabellen können mit Ihren Anmerkungen exportiert werden.

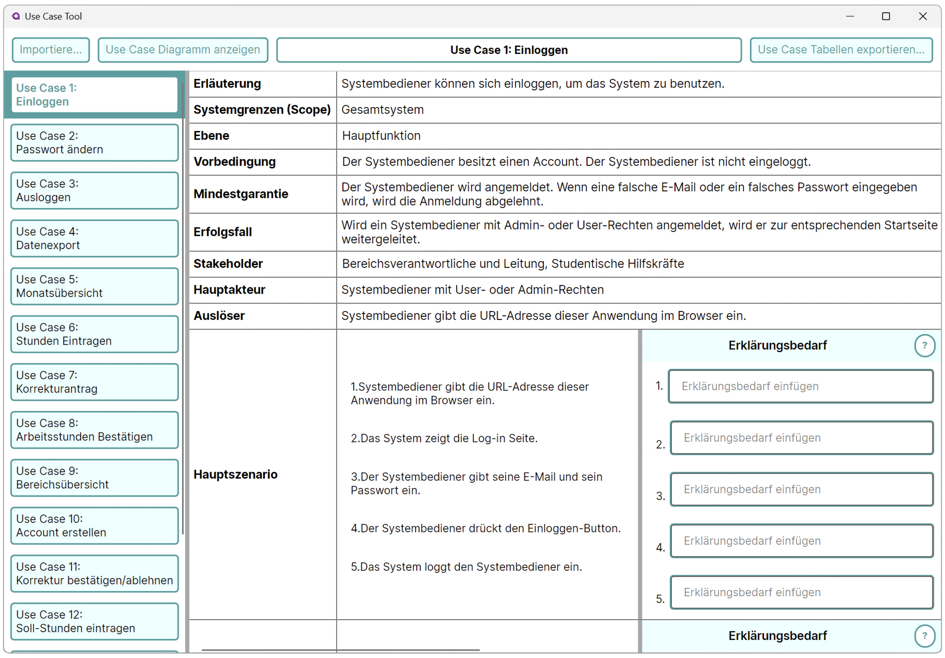
4. Reimport von exportierten Use Case-Tabellen als PDF- und CSV-Datei ist zur Weiterbearbeitung möglich.

Figure C.1: Improved ImportView of the 'Use Case Tool'



Das Dokument wurde erfolgreich gespeichert unter:
"C:/UseCaseExport/UseCaseTabellenArbeitszeittabelle.pdf"

Figure C.2: Improved message box after exporting a file with the 'Use Case Tool'


Use Case Tool
Importiere...
Use Case Diagramm anzeigen
Use Case 1: Einloggen
Use Case Tabellen exportieren...
Use Case 1: Einloggen
Use Case 2: Passwort ändern
Use Case 3: Ausloggen
Use Case 4: Datenexport
Use Case 5: Monatsübersicht
Use Case 6: Stunden Eintragen
Use Case 7: Korrekturantrag
Use Case 8: Arbeitsstunden Bestätigen
Use Case 9: Bereichsübersicht
Use Case 10: Account erstellen
Use Case 11: Korrektur bestätigen/ablehnen
Use Case 12: Soll-Stunden eintragen
Erklärungsbedarf
?
1. Erklärungsbedarf einfügen
2. Erklärungsbedarf einfügen
3. Erklärungsbedarf einfügen
4. Erklärungsbedarf einfügen
5. Erklärungsbedarf einfügen
Erklärungsbedarf
?

Erläuterung	Systembediener können sich einloggen, um das System zu benutzen.
Systemgrenzen (Scope)	Gesamtsystem
Ebene	Hauptfunktion
Vorbedingung	Der Systembediener besitzt einen Account. Der Systembediener ist nicht eingeloggt.
Mindestgarantie	Der Systembediener wird angemeldet. Wenn eine falsche E-Mail oder ein falsches Passwort eingegeben wird, wird die Anmeldung abgelehnt.
Erfolgsfall	Wird ein Systembediener mit Admin- oder User-Rechten angemeldet, wird er zur entsprechenden Startseite weitergeleitet.
Stakeholder	Bereichsverantwortliche und Leitung, Studentische Hilfskräfte
Hauptakteur	Systembediener mit User- oder Admin-Rechten
Auslöser	Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein.
Hauptszenario	1.Systembediener gibt die URL-Adresse dieser Anwendung im Browser ein. 2.Das System zeigt die Log-in Seite. 3.Der Systembediener gibt seine E-Mail und sein Passwort ein. 4.Der Systembediener drückt den Einloggen-Button. 5.Das System loggt den Systembediener ein.

Figure C.3: Improved button in the UseCaseView of the 'Use Case Tool', displayed in its minimum height and width

Appendix D

Contents on the USB Drive

The USB drive supplied with this thesis contains the following:

- The *Git* repository called "ba-denise" that was used to track the development process of the 'Use Case Tool'
- The SRS "Working-time-table.pdf" used for acceptance testing
- A directory called "TestSpecifications" containing eleven SRSs that have been used to test the 'Use Case Tool' during development
- A directory called "UseCaseTool_UsabilityTest", which contains the 'Use Case Tool' in executable form at the time of the usability tests
- A directory called "Study_Evaluation" containing:
    - The results of the SUS questionnaire
    - The calculated results of the SUS questionnaire
    - The anonymized notes taken during the Usability testing
    - The summarized comments and observations based on the notes
- A directory called "DrawnFigures", which contains self-drawn figures and the self-drawn pencil icon
- A directory called "UseCaseTool_improved", which contains the 'Use Case Tool' in executable form after all improvements have been made
- The *LaTeX* archive used to build this document

List of Figures

List of Tables

Acronyms

IEEE Institute of Electrical and Electronic Engineers. 1, 3

MVVM Model-View-ViewModel. 10, 19

NFR Non-Functional Requirement. 1, 3, 7, 15, 27

SRS Software Requirements Specification. 1–4, 7, 11–16, 19, 20, 22, 26–28, 34, 37, 38, 40–42

SU System Usability. 9

SUS System Usability Scale. 8, 9, 29, 31, 32, 35, 41, 42, 53, 55

UI User Interface. 2, 7, 10, 11, 14, 15, 17, 19, 21–25, 33, 34, 38, 41, 42, 53

UML Unified Modeling Language. 6

WPF Windows Presentation Foundation. 10

XAML eXtensible Application Markup Language. 10, 11

Bibliography

[1] P. T. K. Aaron Bangor and J. T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human–Computer Interaction*, 24(6):574–594, 2008.

[2] S. W. Ali, Q. A. Ahmed, and I. Shafi. Process to enhance the quality of software requirement specification document. In *2018 International Conference on Engineering and Emerging Technologies (ICEET)*, pages 1–7, 2018.

[3] M. Aristarán and M. Tigas. Introducing tabula. Available online at `https://source.opennews.org/articles/introducing-tabula/`, accessed on 2024-02-24.

[4] M. Aristarán, M. Tigas, J. B. Merrill, J. Das, D. Frackman, and T. Swicegood. Github: tabulapdf / tabula. Available online at `https://github.com/tabulapdf/tabula`, accessed on 2024-02-24.

[5] AvaloniaCommunity. GitHub: AvaloniaCommunity/MessageBox.Avalonia. Available online at `https://github.com/AvaloniaCommunity/MessageBox.Avalonia`, accessed on 2024-03-06.

[6] AvaloniaUI OÜ. AvaloniaUI: FAQ. Available online at `https://docs.avaloniaui.net/docs/faq`, accessed on 2024-01-23.

[7] AvaloniaUI OÜ. AvaloniaUI on GitHub. Available online at `https://github.com/AvaloniaUI/Avalonia`, accessed on 2024-01-23.

[8] AvaloniaUI OÜ. AvaloniaUI: Welcome. Available online at `https://docs.avaloniaui.net/docs/welcome`, accessed on 2024-01-23.

[9] AvaloniaUI OÜ. Transitioningcontentcontrol. Available online at `https://docs.avaloniaui.net/docs/reference/controls/detailed-reference/transitioningcontentcontrol`, accessed on 2024-03-04.

[10] BobLd. Github: Bobld / tabula-sharp. Available online at `https://github.com/BobLd/tabula-sharp`, accessed on 2024-02-24.

[11] P. Bourque and R. Fairley. *Guide to the Software Engineering Body of Knowledge - SWEBOK V3.0.* IEEE Computer Society, 01 2014.

[12] J. Brooke. Sus: a "quick and dirty usability scale. *Usability evaluation in industry*, 189(3):189–194, 1996.

[13] J. Carifio and R. Perla. Ten common misunderstandings, misconceptions, persistent myths and urban legends about likert scales and likert response formats and their antidotes. *Journal of Social Sciences*, 3, 03 2007.

[14] L. Chazette, W. Brunotte, and T. Speith. Exploring explainability: A definition, a model, and a knowledge catalogue. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*, pages 197–208, 2021.

[15] L. Chazette, W. Brunotte, and T. Speith. Explainable software systems: From requirements analysis to system evaluation. *Requirements Engineering*, 27(4):457–487, dec 2022.

[16] L. Chazette and K. Schneider. Explainability as a non-functional requirement: challenges and recommendations. *Requirements Engineering*, 25(4):493–514, 2020.

[17] A. Cockburn. Basic use case template. *Humans and Technology, Technical Report*, 96:28, 1998.

[18] A. Cockburn. *Writing Effective Use Cases.* Addison-Wesley, 2001.

[19] H. Deters, J. Droste, M. Fechner, and J. Klünder. Explanations on demand - a technique for eliciting the actual need for explanations. In *2023 IEEE 31st International Requirements Engineering Conference Workshops (REW)*, pages 345–351, 2023.

[20] H. Deters, J. Droste, and K. Schneider. A means to what end? evaluating the explainability of software systems using goal-oriented heuristics. In *Proceedings of the 27th International Conference on Evaluation and Assessment in Software Engineering*, EASE '23, page 329–338, New York, NY, USA, 2023. Association for Computing Machinery.

[21] Europäische Kommission. Deutschland: Finanzierung der Hochschulbildung. Available online at `https://eurydice.eacea.ec.europa.eu/de/national-education-systems/germany/finanzierung-der-hochschulbildung`, accessed on 2024-02-20.

[22] A. Fatwanto. Software requirements specification analysis using natural language processing technique. In *2013 International Conference on QiR*, pages 105–110, 2013.

[23] U. Hammerschall and G. Beneken. *Software Requirements* . Pearson Deutschland, 2013.

[24] P. Hsia, D. Kung, and C. Sell. Software requirements and acceptance testing. *Annals of Software Engineering*, 3(1):291–317, 1997.

[25] Institute of Electrical and Electronics Engineers. IEEE guide for software requirements specifications. *IEEE Std 830-1984*, pages 1–26, 1984.

[26] Institute of Electrical and Electronics Engineers. IEEE recommended practice for software requirements specifications. *IEEE Std 830-1998*, pages 1–40, 1998.

[27] Institute of Electrical and Electronics Engineers. ISO/IEC/IEEE international standard - systems and software engineering–vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541, 2017.

[28] Institute of Electrical and Electronics Engineers. ISO/IEC/IEEE international standard - systems and software engineering – life cycle processes – requirements engineering. *ISO/IEC/IEEE 29148:2018(E)*, pages 1–104, 2018.

[29] I. Jacobson. Object-oriented development in an industrial environment. In *Conference Proceedings on Object-Oriented Programming Systems, Languages and Applications*, OOPSLA '87, page 183–191, New York, NY, USA, 1987. Association for Computing Machinery.

[30] I. Jacobson and A. Cockburn. Use cases are essential: Use cases provide a proven method to capture and explain the requirements of a system in a concise and easily understood format. *Queue*, 21(5):66–86, nov 2023.

[31] A. Joshi, S. Kale, S. Chandel, and D. Pal. Likert scale: Explored and explained. *British Journal of Applied Science & Technology*, 7:396–403, 01 2015.

[32] S. Kirk. 10 years of avalonia. Available online at `https://avaloniaui.net/Blog/10-years-of-avalonia`, accessed on 2024-01-23.

[33] K. Konrad. *Lautes Denken*, pages 476–490. Springer Fachmedien Wiesbaden, Wiesbaden, 2020.

[34] J. Kuchta and P. Padhiyar. Extracting concepts from the software requirements specification using natural language processing. In *2018 11th International Conference on Human System Interaction (HSI)*, pages 443–448, 2018.

[35] M. A. Köhl, K. Baum, M. Langer, D. Oster, T. Speith, and D. Bohlender. Explainability as a non-functional requirement. In *2019 IEEE 27th International Requirements Engineering Conference (RE)*, pages 363–368, 2019.

[36] B. Lepri, N. Oliver, E. Letouzé, A. Pentland, and P. Vinck. Fair, Transparent, and Accountable Algorithmic Decision-making Processes. *Philosophy & Technology*, 31(4):611–627, 2018.

[37] R. Likert. *A Technique for the Measurement of Attitudes.* Number Nr. 136-165 in A Technique for the Measurement of Attitudes. Archives of Psychology, 1932.

[38] L. A. MacVittie. *XAML in a Nutshell.* O'Reilly Media, Inc., 2006.

[39] J. McManus. A stakeholder perspective within software engineering projects. In *2004 IEEE International Engineering Management Conference (IEEE Cat. No.04CH37574)*, volume 2, pages 880–884 Vol.2, 2004.

[40] Microsoft. Create .net apps faster with nuget. Available online at `https://www.nuget.org`, accessed on 2024-03-02.

[41] R. Miller and C. T. Collins. Acceptance testing. *Proc. XPUniverse*, 238, 2001.

[42] M. T. Orne. Demand characteristics and the concept of quasi-controls. *Artifacts in behavioral research: Robert Rosenthal and Ralph L. Rosnow's classic books*, 110:110–137, 2009.

[43] M. Osborne and C. MacNish. Processing natural language software requirement specifications. In *Proceedings of the Second International Conference on Requirements Engineering*, pages 229–236, 1996.

[44] J. Pruitt and T. Adlin. *The persona lifecycle: keeping people in mind throughout product design.* Elsevier, 2010.

[45] J. Saldaña. *The Coding Manual for Qualitative Researchers.* Sage, 2nd edition, 2013.

[46] D. Stahlberg and D. Frey. *Einstellungen: Struktur, Messung und Funktion*, pages 219–252. Springer Berlin Heidelberg, Berlin, Heidelberg, 1997.

[47] S. Steiger. GitHub: PdfSharpCore/docs/MigraDocCore/index.md. Available online at `https://github.com/ststeiger/PdfSharpCore/blob/master/docs/MigraDocCore/index.md`, accessed on 2024-03-05.

[48] M. Stonis. *Enterprise Application Patterns Using .NET MAUI*. Microsoft Developer Division, .NET, and Visual Studio product teams, 2022.

[49] G. Sullivan and A. Artino. Analyzing and interpreting data from likert-type scales. *Journal of graduate medical education*, 5:541–542, 12 2013.

[50] W. Tichy. Hints for reviewing empirical work in software engineering. *Empirical Software Engineering*, 5:309–312, 12 2000.

[51] UglyToad. GitHub: UglyToad/PdfPig. Available online at `https://github.com/UglyToad/PdfPig`, accessed on 2024-03-05.

[52] A. Umber and I. S. Bajwa. Minimizing ambiguity in natural language software requirements specification. In *2011 Sixth International Conference on Digital Information Management*, pages 102–107, 2011.

[53] Uno Platform. Create beautiful .net apps faster. Available online at `https://platform.uno`, accessed on 2024-03-02.